University of Alberta

Fast Neural System Identification And Application To Adaptive Processing

by

Won-Kuk Son ©

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Doctor of Philosophy

Department of Electrical and Computer Engineering
Edmonton, Alberta
Spring 1998

National Library of Canada

Acquisitions and Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada

Acquisitions et services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-29110-3

Canada

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Fast Neural System Identification And Application To Adaptive Processing** submitted by **Won-Kuk Son** in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

KE Bollinger

K. E. Bollinger (Supervisor)

H. J. Marquez

Roger W Toogood

R. W. Toogood

Brian Andrews

B. Andrews

Robert E Gander

R. E. Gander (Ext.)

Date: March 25, 1998

In Gratitude To My Parents

Abstract

The design of a fast, accurate, and robust system identification method, and application to adaptive control are the main objectives of this research. This research aim is carried out through black-box modeling system identification using artificial neural networks and an optimal control method. The black-box model implicitly includes the advantage that no physical insight into plant dynamics is available or used in the process of adaptive control design. Applications can include real environments such as rigid-body robot manipulators with unknown friction, payload, backlash, and external disturbances, free-floating robot manipulators with inherently unclear mass-related properties, underwater robotic vehicles with complex hydrodynamic effects, ground vehicles for suspension control with random road conditions, microrobots with viscous effects at low speed, etc.. The black-box modeling is a strategy that puts all uncertain dynamics (including even analytically can-be-known dynamics to overcome the limitation of *ad-hoc* control) into a black-box having only input and output measurements.

System identification based on a black-box model is carried out through a devised recurrent neural network and a suitably modified on-line training algorithm. The algorithm is fast in learning speed, accurate in identification error, and robust with respect to different plants dynamics. Although artificial neural networks (ANNs) have excellent capability, for example function approximation, input/output mapping, massive parallel processing, etc., their learning algorithms have problems of speed, accuracy and robustness generally due to the error back-propagation training strategy. In this thesis, these drawbacks are solved through a novel recurrent neural

topology and modified on-line training algorithm. The developed system identification techniques show excellent performance and take into account uncertain dynamics as well as nominal dynamics simultaneously by inputs and outputs via on-line observation. The adaptive control algorithm based on the black-box identification excludes the possible use of previously can-be-known analytic information about a specific plant in order to prevent *ad hoc* (control) processor working for a specific system only. Therefore, a major goal of this thesis is to design a flexible adaptive control for diverse nonlinear systems and includes the development of powerful neural networks with robust connection weights training algorithms. This research gives specific attention to this aspect.

The neural system identifier, mentioned above, is combined into optimal control techniques for tracking problems of SISO to MIMO systems under the *certainty-equivalence principle*. Since the conventional combination of model-based identifiers and optimal techniques has been carried out using linear models, the use of nonlinear neural identifiers requires a new approach with optimization methods. The overall control scheme developed in this research is categorized as a *multi-variable adaptive self-tuning control with neural identifier*. The control method, supported by a neural black-box model, can cope well for nominal and uncertain dynamics with fast tracking speed and wide operating conditions. It is applicable to different nonlinear systems without changing control schemes. For a benchmark plant, the rigid-body, multi-joint, robot manipulator is used to test the concepts because it exhibits highly nonlinear, strongly coupled MIMO, and fast time-varying elements with uncertain nonlinear dynamics for the tracking purpose via pure input/output measurements.

Acknowledgements

I would like to express my deepest gratitude to Prof. K.E. Bollinger for his supervision, guidance and encouragement (not to mention patience) throughout the duration of this research project. I would also like to thank Dr. H. Marquez and Dr. R.W. Toogood for their help and suggestions as well as agreeing to sit on my advisory committee. I extend my thanks to Dr. B. Andrews in the Department of Biomedical Engineering for his advice and deep interest in this research topic. I also thank Dr. R.E. Gander as a member of my examining committee for his evaluation of this thesis.

In addition to the above, there are many other people to whom I would like to offer my thanks. Among them, my special friend, Alexander Love (Alberta Ltd.) would not be easily forgotten about his friendship. He provided the necessary information and encouragement by sharing my experiences in times of difficulties.

The financial assistance by the Department of Electrical and Computer Engineering, University of Alberta, and the Natural Sciences and Engineering Research Council of Canada (NSERC) is also acknowledged.

And last, but certainly not least, I would like to express my gratitude to my parents and family whose love and trust in me made this work completed.

Contents

1 Overall Introduction **1**

1.1 Introduction 1

1.2 Black-box Neural System Identification 5

1.3 Literature Reviews 7

1.3.1 Neural System Identification and Neuromorphic Controls . . . 8

1.3.2 Non-Neuromorphic Adaptive Robot Control 14

1.3.3 Summary 18

1.4 Objectives Of the Thesis 20

1.5 Thesis Layout 22

Bibliography **25**

2 Robotics For Computer Simulation **31**

2.1 Description of Manipulator by D-H Notation 31

2.1.1 Determination of Coordinate System in Robotics 31

2.1.2 Determination of the Kinematic Parameters 33

2.2 Determination of Transformation Matrix 35

2.2.1 Recursive Transformation Matrix 35

2.2.2 Partition and Decomposition of Transformation Matrix 36

2.3 Orientation of End-Effector 37

2.3.1 An Algorithm for Calculating Ψ_x, Ψ_y, and Ψ_z 38

2.4 Summary - Need for Six-joint Dynamics 41

2.5 Mathematical Modeling of PUMA Manipulator Dynamics 42

2.6 Derivation of L-E Dynamics Model for Motion of Robot Manipulator 43
2.6.1 Lagrange-Euler Equation 44
2.6.2 Kinetic Energy of a Robot Manipulator 45
2.6.3 Potential Energy of a Robot Manipulator 49
2.6.4 L-E Dynamics Equations of a Robot Manipulator 50
2.6.5 Computational Simplification of Dynamics Equation using $Q-$ *matrix* 53
2.7 Customized Closed-form of Dynamics Equation for Six d-o-f PUMA Robot 57
2.7.1 Customizing Dynamics Equation 57
2.7.2 Parameters of PUMA 600 59
2.8 Digital Simulation of PUMA 600 Motion 61
2.8.1 Measured Angular Velocity and Position with Measurement Error 62
2.8.2 About Generation of Gaussian Noise 63
2.9 Remarks 64

Bibliography **65**

3 Artificial Neural Networks **67**
3.1 Introduction 67
3.1.1 Neural Networks Application 70
3.1.2 History of Neural Networks Research 73
3.2 Modeling of A Neuron 76
3.2.1 Structure of Spinal Motor Neuron 76
3.2.2 How Neurons Work 78
3.3 Artificial Models of the Neuron 79
3.3.1 McCulloch-Pitts Neuron 79
3.3.2 Integration Neuron 80
3.3.3 Generic Connectionist Neuron 81
3.4 Neural Networks Classification 84

3.4.1 Feedforward Neural Networks 85
3.4.2 Recurrent Neural Networks 87
3.5 Neural Network Training Algorithm 89
3.5.1 Supervised Training: Backpropagation Algorithm 90
3.5.2 Unsupervised Training 92
3.6 Remarks on ANNs 93

Bibliography **94**

4 Architecture Of Recurrent Neural Networks **99**
4.1 Introduction 99
4.2 Dynamic Properties By Feedback and Memory 100
4.3 Jordan's RNN 101
4.4 Elman's RNN 102
4.5 Proposed SERNN 104
4.6 Architectural Techniques 108
4.6.1 A New Neuron Model Of Error-Feedback 116
4.6.2 A New Feedback Of Supervision Signal 119
4.6.3 Inter-Layer Forward Connections 120
4.7 Schematic Diagram of SERNN 121
4.8 Remarks On the Developed SERNN 124

Bibliography **126**

5 Real-Time Multi-layer Neural Networks Training Algorithm **129**
5.1 Introduction 129
5.2 Error Backpropagation Algorithm 131
5.3 Application of RLS to ANNs Training 132
5.3.1 Least-Squares (LS) Algorithm 133
5.3.2 Standard RLS (SRLS) Algorithm 134
5.4 Modified RLS (MRLS) Algorithm 140

5.4.1 Derivation of Estimates Update Law 140
5.4.2 Derivation of Covariance Matrix Update Law 145
5.4.3 Derivation of Input Vector Update Law 147
5.4.4 Summary of MRLS Algorithm 148
5.4.5 Expected Features Of MRLS Algorithm 149
5.4.6 Design Of New Exponential-Weight Variable Forgetting Factor 149
5.4.7 Periodic Resetting of Covariance Matrix 164
5.5 Application of MRLS to Neural Training 170
5.5.1 Linearization of Neural Model 170
5.5.2 Derivation Of Explicit Supervisory Signal For Hidden Layers . 174
5.6 Remarks On MRLS Algorithm For ANNs Training 182

Bibliography **183**

6 Simulations Of Neural Nonlinear System Identification **186**
6.1 Objectives Of Simulation 186
6.2 Test On A Nonlinear SISO Plant 1 188
6.2.1 Dynamics Of Plant 1 188
6.2.2 Performance Table Of Identification On Plant 1 189
6.3 Test On A Nonlinear SISO Plant 2 190
6.3.1 Dynamics Of Plant 2 190
6.3.2 Performance Table Of Identification On Plant 2 191
6.4 Test On A Nonlinear SISO Plant 3 192
6.4.1 Dynamics Of Plant 3 192
6.4.2 Performance Table Of Identification On Plant 3 193
6.5 Test On A Nonlinear SISO Plant 4 194
6.5.1 Dynamics Of Plant 4 194
6.5.2 Performance Table Of Identification On Plant 4 195
6.6 Test On A Nonlinear MIMO Plant 5 Of Swing Leg 196
6.6.1 Dynamics Of Plant 5 196
6.6.2 Performance Table Of Identification On Plant 5 198

6.7 Conclusions and Simulation Results 199
6.7.1 With Initial Weight Set 1 On Plant 1 202
6.7.2 With Initial Weight Set 2 On Plant 1 204
6.7.3 With Initial Weight Set 3 On Plant 1 206
6.7.4 With Initial Weight Set 4 (Random Number) On Plant 1 . . . 209
6.7.5 With Initial Weight Set 1 On Plant 2 212
6.7.6 With Initial Weight Set 2 On Plant 2 214
6.7.7 With Initial Weight Set 3 On Plant 2 217
6.7.8 With Initial Weight Set 4 On Plant 2 219
6.7.9 With Initial Weight Set 1 On Plant 3 222
6.7.10 With Initial Weight Set 2 On Plant 3 224
6.7.11 With Initial Weight Set 3 On Plant 3 227
6.7.12 With Initial Weight Set 4 (Random Number) On Plant 3 . . . 229
6.7.13 With Initial Weight Set 1 On Plant 4 232
6.7.14 With Initial Weight Set 2 On Plant 4 234
6.7.15 With Initial Weight Set 3 On Plant 4 237
6.7.16 With Initial Weight Set 4 (Random Number) On Plant 4 . . . 239
6.7.17 With Initial Weight Set 1 On MIMO Plant 5 (Swing Leg) . . . 242
6.7.18 With Initial Weight Set 2 On MIMO Plant 5 (Swing Leg) . . . 247
6.7.19 With Initial Weight Set 3 On MIMO Plant 5 (Swing Leg) . . . 251
6.7.20 With Initial Weight Set4 (Random Number) On Plant 5 (Swing Leg) 256
6.8 Remarks on Simulation Results 260

Bibliography **262**

7 Application of Neural System Identification To Adaptive Control Processing **263**
7.1 Introduction 263
7.2 Adaptive Self-tuning Control 266
7.3 Optimal LOQ Controller Design For MIMO System 267

7.4 Simulation Results 272
7.4.1 Without Measurement Noise and Abrupt Dynamic Change . . 273
7.4.2 Addition of Measurement Noise and Abrupt Dynamic Change On Different Trajectories 277
7.5 Remarks 282

Bibliography **284**

8 Conclusions **286**

8.1 Contributions and Findings 287
8.2 Recommendations for Further Research Directions 295

Bibliography **298**

A Customized L-E Dynamics Equation for Three-DOF PUMA Robot 300

List of Tables

2.1 Structural Kinematic Parameters 35

2.2 Kinematic Link Parameters for the PUMA 600 59

2.3 Center-Of-Mass and Relative Link Mass 60

2.4 Radii-of-Gyration 61

6.1 Identification Performance of **Conventional** Methods on **Plant 1** . . 189

6.2 Identification Performance of **Proposed** Methods on **Plant 1** 189

6.3 Identification Performance of **Conventional** Methods on **Plant 2** . . 191

6.4 Identification Performance of **Proposed** Methods on **Plant 2** 191

6.5 Identification Performance of **Conventional** Methods on **Plant 3** . . 193

6.6 Identification Performance of **Proposed** Methods on Plant 3 193

6.7 Identification Performance of **Conventional** Methods on **Plant 4** . . 195

6.8 Identification Performance of **Proposed** Methods on **Plant 4** 195

6.9 Identification Performance of **Conventional** Methods on **Plant 5** . . 198

6.10 Identification Performance of **Proposed** Methods on **Plant 5** 198

List of Figures

1 A System Diagram Of A Black-box Model 5
2 A System With A Known Mathematical Expression 6
3 Functions Of Functions 6

4 PUMA Manipulator with D-H Notation 32
5 Illustration of Structural Parameters 34
6 R-P-Y Angles for Orientation 37
7 A Point $\mathbf{r}_i^i$ in Link i. 45

8 Comparative View of Biological and Artificial Neuron 77
9 McCulloch-Pitts Neuron 80
10 A integrate-and-fire neuron 81
11 Generic Connectionist Neuron 82

12 Jordan's Recurrent Neural Network 102
13 Elman's Recurrent Neural Network 103
14 Proposed Supervision & Error RNN (SERNN) 107
15 From OL-IN To OL-ON and From HL-IN To HL-ON 111
16 From HL-IN To OL-IN For Inter-Layer 111
17 From HL-IN To OL-ON For Inter-Layer 111
18 From OL-ON To OL-IN and From HL-ON To HL-IN 112
19 From OL-ON To HL-ON For Inter-Layer 112
20 From VL-VN(Supervision) To OL-IN 112
21 From VL-VN(Supervision) To HL-ON 113

22 From VL-VN(Supervision) To HL-IN 113
23 From VL-VN(Output-Layer Error) To OL-IN 113
24 From VL-VN to (a) HL-IN and (b) HL-ON 114
25 From VL-VN(Hidden-Layer Error) To HL-IN 115
26 Unity Time-Delay For Memory 115
27 A Conventional Neuron Model 116
28 A New Neuron Model 117
29 A New Neuron Model Observed By Feedback Control 118
30 Real-Time System Identification (Actual Output + Model Output) . . 119
31 Identification Error 119
32 Single-Layer Schematic of SERNN 122
33 Multi-Layer (Detailed) Schematic of SERNN 123

34 Cauchy Function Profile of $x_j(k)$ 154
35 Then $(a):\ x_j(k) = \frac{y_j^{act}(k)}{y_j^{act}(k-1)}$, $(b):\ x_j(k) = \frac{y_j^{act}(k-1)}{y_j^{act}(k)}$ 155
36 Then $(c):\ x_j(k) = \frac{y_j^{act}(k-1)}{y_j^{act}(k)}$, $(d):\ x_j(k) = \frac{y_j^{act}(k)}{y_j^{act}(k-1)}$ 156
37 Then $(e):\ x_j(k) = \frac{[|y_j^{act}(k)|-|y_j^{act}(k-1)|]+y_j^{act}(k)}{y_j^{act}(k)} = 1$,
$(f):\ x_j(k) = \frac{[|y_j^{act}(k-1)|-|y_j^{act}(k)|]+y_j^{act}(k-1)}{y_j^{act}(k-1)} = 1$ 156
38 Then $(g)\&(h):\ x_j(k) = \frac{[|y_j^{act}(k)|-|y_j^{act}(k-1)|]+|y_j^{act}(k-1)|}{w} y_j^{act}(k-1) = \frac{y_j^{act}(k)}{y_j^{act}(k-1)}$ 157
39 Then $(i)\&(j):\ x_j(k) = \frac{[|y_j^{act}(k-1)|-|y_j^{act}(k)|]+|y_j^{act}(k)|}{y_j^{act}(k)} = \frac{y_j^{act}(k-1)}{y_j^{act}(k)}$ 157
40 Then $(k):\ x_j(k) = \frac{1.5 \cdot y_j^{act}(k-1)}{1.0}$, $(l):\ x_j(k) = \frac{1.5 \cdot y_j^{act}(k)}{1.0}$, $(m):\ x_j(k) = \frac{1.0}{1.0}$ 158
41 Single-Layer SERNN With Three Inputs/Outputs 160
42 Actual Outputs of Joint 1,2,3 Identified by Model Outputs 161
43 VFF Profile Used For Joint 1 161
44 VFF Profile Used For Joint 2 161
45 VFF Profile Used For Joint 3 162
46 Identification Error of Joint 1 162
47 Identification Error of Joint 2 162

48 **Identification Error** of **Joint 3** 163
49 Initial Tracking Performances on Joint 1,2,3 163
50 Input (short) and output (tall) for a test plant 167
51 Trace of covariance matrix in output layer 168
52 Trace of covariance matrix in hidden layer 168
53 Plant output identified by conventional and proposed methods 169
54 Output identification error 169
55 Forgetting factor profiles in output layer 169
56 Forgetting factor profiles in hidden layer 170
57 Trace of input vector update of $Q(k)$ for MRLS algorithm 170
58 Linear Observation of A Neuron 173
59 Schematic Diagram of Multi-Layer SERNN 175

60 Input and Output of Nonlinear Plant1 188
61 Input and Output of Nonlinear Plant2 190
62 Input and Output of Nonlinear Plant3 192
63 Input and Output of Nonlinear Plant4 194
64 Two Outputs of Nonlinear MIMO Plant 5 of Swing Leg 196
65 Two Inputs of Nonlinear MIMO Plant 5 of Swing Leg 196
66 Actual output **identification** by model output 202
67 **Identification error** between actual and model outputs 202
68 Constant and Variable **Forgetting Factors** in **output layer** 202
69 Constant and Variable **Forgetting Factors** in **hidden layer** 203
70 **Trace of covariance matrices** in **output layer** 203
71 **Trace of covariance matrices** in **hidden layer** 203
72 Sum of Q Elements 204
73 Actual output **identification** by model output 204
74 **Identification error** between actual and model outputs 204
75 Constant and Variable Forgetting Factors in output layer 205
76 Constant and Variable Forgetting Factors in hidden layer 205

77 Trace of covariance matrices in output layer 205
78 Trace of covariance matrices in hidden layer 206
79 Sum of Q Elements 206
80 Actual output **identification** by model output 206
81 **Identification error** between actual and model outputs 207
82 Constant and Variable Forgetting Factors in output layer 207
83 Constant and Variable Forgetting Factors in hidden layer 207
84 Trace of covariance matrices in output layer 208
85 Trace of covariance matrices in hidden layer 208
86 Sum of Q Elements 208
87 Actual output **identification** by model output 209
88 **Identification error** between actual and model outputs 209
89 Constant and Variable Forgetting Factors in output layer 209
90 Constant and Variable Forgetting Factors in hidden layer 210
91 Trace of covariance matrices in output layer 210
92 Trace of covariance matrices in hidden layer 210
93 Sum of Q Elements 211
94 Actual output **identification** by model output 212
95 **Identification error** between actual and model outputs 212
96 Constant and Variable Forgetting Factors in output layer 212
97 Constant and Variable Forgetting Factors in hidden layer 213
98 Trace of covariance matrices in output layer 213
99 Trace of covariance matrices in hidden layer 213
100 Sum of Q Elements 214
101 Actual output **identification** by model output 214
102 **Identification error** between actual and model outputs 214
103 Constant and Variable Forgetting Factors in output layer 215
104 Constant and Variable Forgetting Factors in hidden layer 215
105 Trace of covariance matrices in output layer 215
106 Trace of covariance matrices in hidden layer 216

107 Sum of Q Elements 216
108 Actual output **identification** by model output 217
109 **Identification error** between actual and model outputs 217
110 Constant and Variable Forgetting Factors in output layer 217
111 Constant and Variable Forgetting Factors in hidden layer 218
112 Trace of covariance matrices in output layer 218
113 Trace of covariance matrices in hidden layer 218
114 Sum of Q Elements 219
115 Actual output **identification** by model output 219
116 **Identification error** between actual and model outputs 219
117 Constant and Variable Forgetting Factors in output layer 220
118 Constant and Variable Forgetting Factors in hidden layer 220
119 Trace of covariance matrices in output layer 220
120 Trace of covariance matrices in hidden layer 221
121 Sum of Q Elements 221
122 Actual output **identification** by model output 222
123 **Identification error** between actual and model outputs 222
124 Constant and Variable Forgetting Factors in output layer 222
125 Constant and Variable Forgetting Factors in hidden layer 223
126 Trace of covariance matrices in output layer 223
127 Trace of covariance matrices in hidden layer 223
128 Sum of Q Elements 224
129 Actual output **identification** by model output 224
130 **Identification error** between actual and model outputs 224
131 Constant and Variable Forgetting Factors in output layer 225
132 Constant and Variable Forgetting Factors in hidden layer 225
133 Trace of covariance matrices in output layer 225
134 Trace of covariance matrices in hidden layer 226
135 Sum of Q Elements 226
136 Actual output **identification** by model output 227

137 **Identification error** between actual and model outputs 227
138 Constant and Variable Forgetting Factors in output layer 227
139 Constant and Variable Forgetting Factors in hidden layer 228
140 Trace of covariance matrices in output layer 228
141 Trace of covariance matrices in hidden layer 228
142 Sum of Q Elements 229
143 Actual output **identification** by model output 229
144 **Identification error** between actual and model outputs 229
145 Constant and Variable Forgetting Factors in output layer 230
146 Constant and Variable Forgetting Factors in hidden layer 230
147 Trace of covariance matrices in output layer 230
148 Trace of covariance matrices in hidden layer 231
149 Sum of Q Elements 231
150 Actual output **identification** by model output 232
151 **Identification error** between actual and model outputs 232
152 Constant and Variable Forgetting Factors in output layer 232
153 Constant and Variable Forgetting Factors in hidden layer 233
154 Trace of covariance matrices in output layer 233
155 Trace of covariance matrices in hidden layer 233
156 Sum of Q Elements 234
157 Actual output **identification** by model output 234
158 **Identification error** between actual and model outputs 234
159 Constant and Variable Forgetting Factors in output layer 235
160 Constant and Variable Forgetting Factors in hidden layer 235
161 Trace of covariance matrices in output layer 235
162 Trace of covariance matrices in hidden layer 236
163 Sum of Q Elements 236
164 Actual output **identification** by model output 237
165 **Identification error** between actual and model outputs 237
166 Constant and Variable Forgetting Factors in output layer 237

167 Constant and Variable Forgetting Factors in hidden layer 238
168 Trace of covariance matrices in output layer 238
169 Trace of covariance matrices in hidden layer 238
170 Sum of Q Elements 239
171 Actual output **identification** by model output 239
172 **Identification error** between actual and model outputs 239
173 Constant and Variable Forgetting Factors in output layer 240
174 Constant and Variable Forgetting Factors in hidden layer 240
175 Trace of covariance matrices in output layer 240
176 Trace of covariance matrices in hidden layer 241
177 Sum of Q Elements 241
178 Actual output1 **identification** and **error** by model output1 242
179 Actual output2 **identification** and **error** by model output2 242
180 Constant and Variable Forgetting Factors at output1 in output layer 243
181 Constant and Variable Forgetting Factors at output2 in output layer 243
182 Constant and Variable Forgetting Factors at output1 in hidden layer 243
183 Constant and Variable Forgetting Factors at output2 in hidden layer 244
184 Trace of covariance matrices at output1 in output layer 244
185 Trace of covariance matrices at output2 in output layer 244
186 Trace of covariance matrices at output1 in hidden layer 245
187 Trace of covariance matrices at output2 in hidden layer 245
188 Sum of Q elements (proposed) in *output layer* 246
189 Sum of Q (proposed) elements in *hidden layer* 246
190 Actual output1 **identification** and **error** by model output1 247
191 Actual output2 **identification** and **error** by model output2 247
192 Constant and Variable Forgetting Factors at output1 in output layer 248
193 Constant and Variable Forgetting Factors at output2 in output layer 248
194 Constant and Variable Forgetting Factors at output1 in hidden layer 248
195 Constant and Variable Forgetting Factors at output2 in hidden layer 249
196 Trace of covariance matrices at output1 in output layer 249

197 Trace of covariance matrices at output2 in output layer 249
198 Trace of covariance matrices at output1 in hidden layer 250
199 Trace of covariance matrices at output2 in hidden layer 250
200 Sum of Q elements (proposed) in *output layer* 250
201 Sum of Q elements (proposed) in *hidden layer* 251
202 Actual output1 **identification** and **error** by model output1 251
203 Actual output2 **identification** and **error** by model output2 252
204 Constant and Variable Forgetting Factors at output1 in output layer . 252
205 Constant and Variable Forgetting Factors at output2 in output layer . 252
206 Constant and Variable Forgetting Factors at output1 in hidden layer . 253
207 Constant and Variable Forgetting Factors at output2 in hidden layer . 253
208 Trace of covariance matrices at output1 in output layer 253
209 Trace of covariance matrices at output2 in output layer 254
210 Trace of covariance matrices at output1 in hidden layer 254
211 Trace of covariance matrices at output2 in hidden layer 254
212 Sum of Q elements (proposed) in *output layer* 255
213 Sum of Q elements (proposed) in *hidden layer* 255
214 Actual output1 **identification** and **error** by model output1 256
215 Actual output2 **identification** and **error** by model output2 256
216 Constant and Variable Forgetting Factors at output1 in output layer . 257
217 Constant and Variable Forgetting Factors at output2 in output layer . 257
218 Constant and Variable Forgetting Factors at output1 in hidden layer . 257
219 Constant and Variable Forgetting Factors at output2 in hidden layer . 258
220 Trace of covariance matrices at output1 in output layer 258
221 Trace of covariance matrices at output2 in output layer 258
222 Trace of covariance matrices at output1 in hidden layer 259
223 Trace of covariance matrices at output2 in hidden layer 259
224 Sum of Q elements (proposed) in *output layer* 259
225 Sum of Q elements (proposed) in *hidden layer* 260

226 Adaptive Self-tuning Scheme 267
227 Adaptive Self-Tuning Scheme Based on Neural Identification 271
228 Joint1 Position Tracking and Error 273
229 Joint2 Position Tracking and Error 273
230 Joint3 Angular Position Tracking and Error 274
231 Joint1 Angular Velocity Tracking and Error 274
232 Joint2 Angular Velocity Tracking and Error 275
233 Joint3 Angular Velocity Tracking and Error 275
234 Joint1 Angular Velocity System Identification and Identification Error 276
235 Joint2 Angular Velocity System Identification and Identification Error 276
236 Joint3 Angular Velocity System Identification and Identification Error 276
237 Joint1 Position Tracking and Error 278
238 Joint2 Position Tracking and Error 278
239 Joint3 Angular Position Tracking and Error 279
240 Joint1 Angular Velocity Tracking and Error 279
241 Joint2 Angular Velocity Tracking and Error 280
242 Joint3 Angular Velocity Tracking and Error 280
243 Joint1 Angular Velocity System Identification and Identification Error 281
244 Joint2 Angular Velocity System Identification and Identification Error 281
245 Joint3 Angular Velocity System Identification and Identification Error 281

Chapter 1

Overall Introduction

1.1 Introduction

This thesis presents the development of fast, accurate and robust neural system identification techniques and applications to adaptive control. Some complicated dynamic systems such as a robot manipulator, human neuromuscular system and flying vehicle (aircrafts, missiles, etc.) include lots of dynamic complexities. Typical difficulties include nonlinearity, strong coupling effects, time-variance, parameter uncertainty, unmodeled dynamics, etc.. These complexities are difficult to model mathematically. For the adaptive (control) processing to handle these undesirable problems, the behavior (position, velocity, orientation, etc.) of these systems must be able to be interpreted on-line (sometimes in advance) to manage them properly. This motivates the search for a nonlinear system identification algorithm which in the past has been tackled mainly by off-line algorithms or mathematical models. In the past three decades major advances have been made in adaptive identification and control for identifying and controlling linear time-invariant plants with unknown parameters. The choice of the identifier and controller structures was based on well established results in linear system theory. In this research, the first effort is put on the development of a powerful neural identifier performing on-line.

The adaptive system identifier adjusts itself by causing its output to match that of the unknown system, generally to cause its output to be a best least-squares fit to that of the unknown system. Upon convergence, the structure and parameter values of the adaptive system identifier may or may not resemble those of the unknown system, but the input-output response relationships will match. In this sense, the adaptive system identifier becomes a model of the unknown system. New control strategies can be investigated to handle highly complex systems based on a model. One such method that has emerged to deal with these complex nonlinear systems is the use of artificial neural networks (ANNs). It is generally found that ANNs handle the nonlinear identification problem well, although some difficulties for more complex systems still need to be resolved.

In practice, the iterative on-line identification and its adaptive control processing is inevitable to compensate various nominal dynamics and uncertainties through training a ANN model. In order to carry out this objective, first of all, the system identification should *robustly* be achieved based on an adequate, but universal system model. This is essential in order to take into account various dynamic effects in real-life systems by means of only input/output measurements. For example, the adaptive explicit control scheme consists of two separate main blocks of *plant identification* to characterize dynamic effects into a model and *controller gain adjustment* to minimize the control error between desired and actual signals. The author strongly believes that plant identification plays a very important role for controlling nonlinear time-varying system with uncertainties. This is also emphasized in the following statements:

> *The Achilles heel of its self-tuning version (adaptive explicit control scheme), however, is the identification algorithm which was originally introduced more as an afterthought than as an integral part of the design* [SMS91].
>
> *A key part of the self-tuning controller is the identifier* [CG89].

The identification block in this research includes the developed recurrent neural network architecture, and its fast, accurate and robust learning algorithm. The identification task can be separated into four successive subproblems as follows:

- Creating data sets on-line or off-line.
- Building a model (architecture).
- Determining (estimating or training) parameters of the model.
- Testing or diagnosing performances of speed, accuracy, robustness, generality, etc..

The description for step 1 is covered in Chapter 2, while the development and findings for step 2, 3, and 4 are covered in Chapters 4, 5, and 6 respectively.

After the nonlinear system dynamics is identified on-line through the neural network model and efficient training algorithms, its applicability is shown through the adaptive self-tuning (explicit) control scheme for the tracking problem of the robot manipulator presented in Chapter 7. Chapter 7 shows the development of the optimal LOQ (linearly observed quadratic) control based on a fast, accurate and robust neural identifier developed in the previous chapters, and presents simulation results for tracking control without using pre-information of the system being controlled. If a certain control strategy does not require pre-knowledge of the system being concerned, the system identifier will play a more important role in modern control methodology. This is because an exact mathematical modeling of many dynamic systems is difficult to obtain if not impossible in some cases. As systems get more complicated, cost and time are key features in effecting appropriate design strategy. One such example may be found in the *Functional Neuromuscular (or Electrical) Stimulation* of biomedical engineering, where the exact mathematical modeling of human body is almost impossible. There exists potential advantages in using neural networks from application to application. Some real-life stories of neural nets applications are outlined below:

1. Ford Research Lab. (Dearborn, Mich.): developed the road-tested, low-cost, U.S.-made controller that meets the tough new *Clean Air Act* standard for ultra-low vehicle emissions. **Crucial to this success was the ability of advanced neural networks to minimize such parameters** as pollution

and energy overuse, accounting directly and effectively for nonlinearities and noise, in a highly dynamic system. (Also important was the availability of low-cost, high-throughput chips.)

2. Accurate Automation Corp.[1] (Chattanooga, Tenn.): developed the aircraft stabilizer (controller) by using the inputs and outputs of the players who could stabilize the craft under a wide variety of conditions, and then using *neural networks* to replicate these players' behavior (which is an identification of craft-players' behavior off-line). The **neural networks control is a central part of all the efforts**. For this, the designers took few weeks with very little expenditure. They also used neural optimization methods for solving the *mass ratio* problem of the National Aerospace Plane.

3. NASA's Ames Research Center (Moffett Field) and Dryden Flight Research Center (Edwards, both in Calif.): developed the automatic landing system for a real MD-11 jumbo jet. Their first-generation controller was a hybrid of a *neural network* and a traditional control system, in which the neural network had earlier been trained **off-line** to respond to emergencies as experienced pilots do. A second-generation system based on **on-line** learning is in development.

4. Other: The list of applications of *neural networks* is impressive. For example, the neural networks are used in about half the optical character recognition systems on the market today[2].

Meanwhile, neural identification techniques have a wide potential range of applications such as in antenna array processing (blind identification[3]), communication channel equalization, in multiuser detector of CDMA (code division multiple access)[4] communication, in image reconstruction and restoration, in factor analysis, in integrated circuit testing and diagnosis, in designing voice controlled machines, in medical science, etc. as shown in [HJ86], [CCBG90], [JH91], [TLSH91], [Car91], [CJH91], [Sor91], [CPA91], [VM88], [Vac91], [CM92], [KM96], [TS96], and [CSB96].

---

[1] The largest prime contractor in the continuing U.S. efforts in hypersonics.

[2] IEEE Spectrum, vol.35, no.1, Jan., 1998

[3] defined as the problem of separating and estimating multiple source signals from an array of sensors without knowing the characteristic of the transmission channels.

[4] Also called SSMA (spread spectrum multiple access)

1.2 Black-box Neural System Identification

The key points in neural system identification are to design a *robust and general model structure* as a *black-box*, and then to develop a fast, accurate and robust neural training algorithm for various dynamic nonlinear systems. An *architectural model* is imitating the behavior of physical dynamic systems which may be regarded as an unknown black-box having one or more inputs/outputs, while a *mathematical model* contains detailed information of the system under consideration. A black-box is a mechanism, or a device, that accepts an input(s), and produces an output(s). Historically, the word *black-box* comes from the fact that many electronic devices are actually housed in black boxes. The word 'black' calls attention to the fact that we may not know how the internal mechanism converts the inputs into outputs as shown in Figure 1. A certain input-output system may be so complicated that analysis of its behavior is difficult, or impossible, because of the unstructured uncertainty, especially in a nonlinear system. The behavior of the device may vary with time. In the cases such as this, the designer assumes such a system to be a black-box.

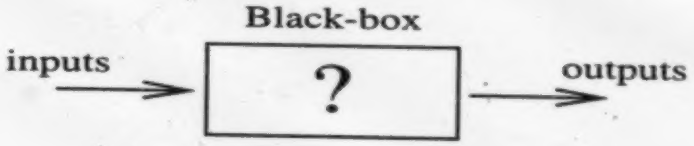


Figure 1: A System Diagram Of A Black-box Model

Figure 2 presents a system with the exact known function $g(\cdot)$. The black-box system identifier associates outputs with inputs without knowing the function $g(\cdot)$ even after the identification process.

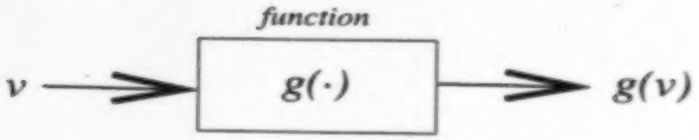


Figure 2: A System With A Known Mathematical Expression

Boxes can be hooked up to one another as shown in Figure 3. This corresponds to the process of forming functions of functions.

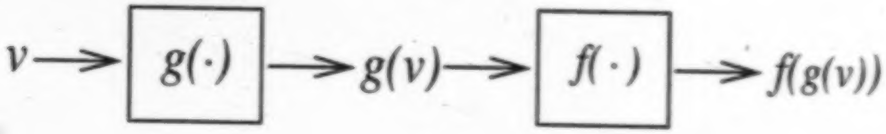


Figure 3: Functions Of Functions

The concept of functional boxes is similar to one in the coding of problems for artificial neural networks. One functional box in Fig. 2 corresponds to a single-layer neural network which forms a conceptual unit and which actually can be available through training the networks. Cascaded boxes in Fig. 3 correspond to multi-layer neural networks. For the recent adaptive control area, one of the major problems associated with the system model for identification is that of studying their behavior so as to predict the outputs corresponding to any particular inputs under unknown environments. Therefore, black-box models not using prior knowledge can have important advantages. These include compensation of the mathematically unmodeled dynamics and the uncertainties, as well as flexibility and applicability for various systems if unknown functions are known through neural networks and training methods in terms of mapping. For the purpose of comparison, it is necessary to distinguish a type of system model that roughly depends on three levels of prior knowledge.

1. **White-box model**: This is the case when a model is perfectly known; it has been possible to construct it entirely from prior knowledge and physical insight. This naturally leads to an *ad-hoc* controller[5] for a plant where prior knowledge

---

[5] Designed for a specific plant and losing the general application.

was obtained.

2. **Grey-box model**: This is the case where some physical insight is available, but several parameters remain to be determined from observed data. It is useful to consider two subcases:

    (a) *Physical modeling*: A model structure can be built on physical grounds, which has a certain number of parameters to be estimated from data. This could, for example, be a state-space model of a given order and structure.

    (b) *Semiphysical modeling*: Physical insight is used to suggest certain non-linear combinations of measured data signal. These new signals are then subjected to model structures of black-box character.

    Most adaptive controllers are classified into this latter modeling method. This method cannot overcome the category of *ad-hoc* control.

3. **Black-box model**: No physical insight is available or used, but the chosen model structure belongs to families that are known to have good flexibility and have been successful in the past, for example, recurrent neural networks for dynamic systems, and feedforward neural networks for static (time-invariant) mapping. Some of the purely adaptive indirect control schemes use this type of system model.

In this thesis, the strategy of the black-box neural system identification is tackled, which requires no prior knowledge about the system under study. This can theoretically result in a pure-measurement based adaptive processor in its application, which will be useful for complex systems where analysis is intractable.

1.3 Literature Reviews

The contents of this review are related to areas of the robotics, artificial neural networks, system identification and optimal control. Therefore, the literature reviews may follow somewhat interdisciplinary subjects.

1.3.1 Neural System Identification and Neuromorphic Controls

There has been considerable interest in learning in the form of neural networks models. The overall complexity of many nonlinear systems and the ideal of a truly general system control have led to much discussion of the use of neural networks. Several neural network models and learning schemes were applied to robot dynamics. A main distinction between these methods is the amount of pre-knowledge about dynamic systems which is used in the design procedure. In this context, the term *pre-knowledge* is related to the characteristic and structure of the analytical form of a system (robot) mathematical model.

[KFS87]and [KUIS88] used the complete available information about the robot dynamic model in the design procedure. This approach uses a neural structure based on non-recurrent single-layer feedforward neural network. They have a deterministic nature, but there are several drawbacks related to poor generalization properties and to the inherent complexity of the implementation of a complete model of robot dynamics.

[CBG90] presented nonlinear system identification using two-layer feedforward neural network in batch and recursive modes. This research was motivated by the theoretical basis of [Cyb89] that neural networks can uniformly approximate any continuous function. The main interest of [CBG90] is to improve the performance of the BP algorithm using the RPE (recursive prediction error) algorithm where the updating rule also can be regarded as an approximation of the *steepest-descent* (gradient) algorithm. They showed that the RPE algorithm is better than the classical BP algorithm in speed through simulations on a single-input single-output system. However, there exist similar drawbacks of the BP algorithm due to the same gradient search and error-propagation down methods. Most of the gradient search methods suffer from problems of limited weights initialization, slow learning speed with free initial weights and local minima. It is questionable that the feedforward neural network they used would work for other dynamic nonlinear systems. The feedforward networks are generally known to have the capability of static mapping.

[ST92] presented a fast training algorithm for multi-layer perceptrons using the combination of the Kalman filter and BP algorithm as an alternative to the backpropagation (BP) algorithm. They have improved the number of iterations by their training algorithms about 20% compared to the performance of the pure-BP algorithm. Their algorithm was tested for pattern classification and pattern recognition problems (static mapping task) through simulated training patterns. Some contributions noticed were: (a) To improve the training speed, they adopted Kalman filter techniques. (b) The separation of each neuron into linear and nonlinear portions was observed for the Kalman filter which is applicable to a linear parametric model. But they failed to apply the Kalman filter method to both output and hidden layers training due to the absence of a hidden-layer supervisor. They expediently used the modified error BP algorithm with the Kalman gain for the hidden-layer training. Their methods can still be affected by shortcomings of the BP algorithm. The important subjects of the *numerical wind-up* in the covariance matrix and the forgetting factor selection were not covered in the Kalman filter application. However, one of their conclusive comment was that the adaptive nature of the Kalman gain makes their algorithms much less likely to get caught in a state other than the *global minimum*, and this is very noteworthy. This is because the *local minimum* problem has been indicated as the most serious defect in the BP training clone. They used the Kalman filter just for the output-layer training. This raises the expectation that the application of the Kalman filter family[6] to both hidden and output layers training may solve the crucial local minimum problem. But application of a pure Kalman filter algorithm to the multi-layer neural nets requires derivation of an explicit teaching signal of hidden-layers.

[ML93b] used two Adalines of neural networks in order to solve for dynamic uncertainties. The use of one Adaline becomes possible from the pre-analysis of the linear factorization method on nonlinear rigid-body dynamics based on original works in [KK85]. In this approach, one Adaline was used with off-line training by the LMS algorithm for structured (nominal) dynamics compensation, while another Adaline used an on-line weight updating mechanism using Lyapunov's method to compensate uncertainties. However, such a linear pre-factorization to derive the mathematical linear dynamics is possible only when physical uncertainties are just on the mass properties of the individual links. Therefore, when dynamic uncertainties exist other than mass property, the control scheme is violated seriously. Two separated Adaline schemes do not make sense under diverse uncertainties because the linear factorization is not possible. Training algorithms using hill-climbing (e.g., gradient descent algorithm like LMS) generally suffer from local minima problems and slow training speed as presented in [MD93]. These become more serious especially when the neural network complexity is increased for a more dexterous robot. The sensitivity to the learning rate for the LMS algorithm causes another problem. The linear factorization requires huge pre-derivation processes when its application is extended to the six d.o.f robot rather than a simple two d.o.f planar model as its simulation. It is definitely required to develop the computerized factorization method for the extended application.

---

[6] The RLS (recursive least-squares) algorithm is one case of the Kalman filter.

[JH95] used techniques of the computed torque method with a PD controller and a neural network in order to improve the poor performance of a robot controller about dynamic uncertainties. This is to compensate the uncertain dynamics on-line in the Cartesian space. To test the capability of uncertainty compensation, Coulomb friction and viscous friction torques have been added to each joint and the payload of $10Kg$ to the third link mass. In particular, this research examined the system performance depending on the location of the neural network controller through simulation scenarios. They exploit the fact that smaller hidden layer signals are helpful for a neural network to capture nonlinearities, which will be useful for more effective modeling. They made use of the two-layer feedforward neural network with a nonlinear hidden-layer and a linear output-layer. The input buffer keeps two sampling periods of old signals with a total of 81 synaptic weights. This control scheme, however, requires heavy calculations for a $n \times n$ inertia matrix, $n \times 1$ Coriolis/centripetal vector, $n \times 1$ gravitational vector and $n \times n$ Jacobian matrix with exact kinematic parameter values (where n is the number of degree-of-freedom). The BP training algorithm they used can suffer from some shortcomings. The pre-compensation of nonlinear terms by the computed torque method is impractical for a high d.o.f. robot controller.

In [LP96] a similar strategy of multi-layer neural networks training to that of [ST92] was described for the system identification. The BP algorithm and Kalman filter were used for hidden and output layers respectively. In order to improve the speed and stability of the BP algorithm, they used the same steepest-descent search and an optimal learning rate different from that (constant) of [ST92]. The optimal learning rate has a value in the range of $[0, 10]$, but they did not show a clear optimization methodology to determine its value. The Kalman filter with the unity forgetting factor that they used may not work in more general problems under different systems and operating conditions. No investigation was shown about burst phenomenon in the Kalman filter. For a dynamic nonlinear system identification, they introduced the time-delay unit (memory) and feedback to input/output signals, which might make a contribution to the learning speed in simulations. In overall performance, the distinct differences are seldom noticed between [ST92] and [LP96] since both research used the BP algorithm for the hidden-layer training. Meanwhile, their emphasis is noticeable that system identification is usually the first step taken by adaptive control engineers since control theory requires an understanding of a system before we try to control it.

[YB89], [OSF$^+$91], [KV92], [BGC96] tried the grey-box model based neural control. In these cases, neural networks can be used as a general computational model. Although the ideal of a pure neural network approach without knowledge about robot dynamics is very promising and has lots of advantages, it is noticed that these approaches might not be very practical. This is due to high dimensionality of input/output spaces, huge training cycles in off-line phase, non-robustness (also speed and accuracy) of learning algorithms, and less generalization properties of overall algorithms. Therefore, the development of a (more) general neural model (architecture) and its robust training algorithm on-line is inevitable for the more black-box model based adaptive signal processing to cover those problems.

[SS95] employed the direct adaptive tracking control design with neural networks based on the original work of [SS92]. This research was extended to classes of multivariable mechanical systems including robot manipulators, where bounds of asymptotic tracking errors and the convergence rate to these bounds are developed. This approach is distinguished from the late 1980s linearly factorized adaptive control schemes. The rigid body motion dynamics is decomposed into a matrix of *unknown* nonlinear functions which is multiplied by a vector of analytic (known) signals. This is carried out based on the agreement that an explicit linear factorization of mechanical systems dynamics is impossible. This is in contrast to previous adaptive control research of [SL86, SL87a, SL87b, SL88, SV89, OS89, SO90, Spo92] which all are based on the linear factorization method. This new adaptive approach learns unknown nonlinear functions including uncertainty through a neural network and simultaneously computes known vectors from the mathematical analysis of motion dynamic equations. This is different from previous approaches where nonlinear component functions must be computed by the analytic factorization process. The previous approaches resulted in the neglect of uncertainty effects except for the uncertainty on the mass property. The new approach has more flexibility for compensating for nonlinear properties including the uncertainty by means of Gaussian radial basis function networks and factorization information. This research shows good findings and results for the stability proof under assumptions of boundedness for states, smooth desired trajectories with up to the second derivatives and an understanding of the state dependence of dynamics in advance, but not the exact functional form. To prove its final performance, the simulation presents the tracking results of a planar two-joint robotic manipulator based on the proposed neural network with a total of 437 nodes and 5224 weights being trained. This neural size seems to be impractical for a real-life application.

[KV95] presented the application of neural networks and fixed gain PID control to the robot control based on the decentralized control scheme originated from [VSK85]. This research used the PID control with manually adjusted gains for a nominal control part and several multi-layer feedforward neural networks for a variable control block. Some features included: (a) They decomposed one big feedforward ANN into several small nets whose outputs then were combined in parallel. The network decomposition was performed based on coordinate information of Lagrange-Euler robot dynamics to reduce the load of neural training algorithm. (b) For a fast training method, the classical RLS (recursive least-squares) algorithm with constant forgetting factor was used to circumvent the drawbacks of the BP (backpropagation) training algorithm. Some shortcomings include: (a) They used a feedforward neural network, which is related more to static mapping, for a dynamic nonlinear input/output mapping. It is well known that recurrent neural networks of feedback and memory are more suitable for dynamic mappings. (b) They did not pay attention to the *numerical wind-up* phenomenon in using the RLS algorithm, which was known as a crucial drawback. (c) They did not present a reasonable teaching signal of the hidden-layer for the RLS algorithm, but used the error backpropagation for a hidden-layer teacher. (The BP algorithms became popular due to the hidden-layer training methodology although there exist some drawbacks.) To cure the slow property of the BP algorithm, they had to carry out previous off-line training for the synaptic weights initialization although the overall control scheme was an on-line strategy.

[BGC96] described the design of a neuro-adaptive trajectory tracking controller for the robot manipulator. The proposed control scheme consists of two modules of feedforward RBFN (radial basis function network) for a system identifier and inversion of its neural emulator. The overall scheme comprises the adaptive indirect control. They used the standard RLS algorithm for the RBFN training and the Kalman filter for the inversion of the neural emulator motivated by [IST92] and [ST92]. The use of the Kalman filter for the control law on-line is noteworthy. Since the popular BP (backpropagation) algorithm is prone to local minima trap, sensitivity to learning parameters and relatively slow in convergence, they used the RLS algorithm for the RBFN training. However, the following disadvantages were noticed:

(a) They did not use the RLS algorithm as an on-line method, but for off-line purpose because the standard RLS clone is not robust enough for an arbitrary neural training data set. A network trained off-line by differently scaled data sets can not take into account time-variance and dynamic uncertainties accurately. (Actually, the RLS algorithm was suitably developed for a time-varying system.) (b) They failed to use the multi-layer neural networks because of the absence of adequate training method without using the BP algorithm. (Although the RBFN has two-layer neural network architecture, it cannot be regarded as a multi-layer neural network because synaptic weights in the hidden-layer are of all unity values. Neurons in the output-layer of the RBFN do not have a nonlinear processing unit and a recurrent structure either.) In order to partly resolve less robustness of the RLS algorithm, they used the off-line training data sets which should be modified by normalizing and scaling differently and manually. But this cannot be a true solution to those problems in the RLS clone. (c) They used the constant forgetting factor of unity. This can cause a problem for algorithmic generalization. (d) They did not cover the topic of the persistent excitation problem for the RLS algorithm. This is one of the important factors in conjunction with the application of the RLS clone. As a side effect of their strategy, the overall control scheme has caused the generalization problem leading to large tracking error when different desired signals are applied. Intensive research to improve the performance of the RLS algorithm is required.

1.3.2 Non-Neuromorphic Adaptive Robot Control

The non-ANNs based adaptive controls of robot manipulators are reviewed below.

[DD79] proposed a simple model reference adaptive control for control of mechanical manipulators. They took into account the effect of payload by combining it with the final link. A Model-Reference-Adaptive Control(MRAC) law was devised using the steepest descent method for a manipulator with counterbalance in order to handle non-linearities and payload. Coupling between joints of the manipulator was neglected and a numerical simulation study was performed with three-joint dynamics.

[TA81] developed a MRAC law which consists of feedforward control and feedback control. The feedforward control reduces the effects of gravity, while feedback control compensates for the position errors, velocity errors, constant disturbances, and acceleration requirements based on the Lyapunov direct method. They assumed low-speed motion to make it possible to neglect Coriolis and centrifugal force, and their simulation was shown with a four-joint manipulator.

[KG83] proposed an adaptive self-tuning controller based on discrete linear time-invariant decoupled model. The controller algorithm assumes that the interaction forces among the joints are negligible. Thus the assumption of slowly varying parameters is unavoidable. Tracking results at a high velocity are not shown. Robustness of the proposed algorithm is questionable for different desired trajectories because the non-linearity ~~in the dynamics of the~~ manipulator strongly depends on the desired trajectories of high velocity profile.

[LC84, LC85] suggested an adaptive perturbation control scheme composed of a nominal control and a variational control. Since nominal control uses the direct calculation of manipulator inverse dynamics along the desired trajectory, it requires full information for the dynamics of the manipulator. The variational control regulating the perturbation with respect to the desired trajectory was based on a linear perturbation model of the manipulator together with a recursive least-squares identification algorithm and a one-step-ahead optimal control algorithm. They showed by computer simulation that their control law was insensitive to variation of payload, but convergence of the control law was not shown explicitly. Their simulation was tested with a three-joint manipulator.

[CHS86] presented an adaptive computed torque or inverse dynamics method for the control of manipulators with rigid links. They tried to show a globally stable control scheme and conditions for parameter convergence as well as its asymptotic properties. However, drawbacks exist in practice due to the fact that the global stability of this method depends on having exact dynamic models and full knowledge of parameters of the system, accuracy, and speed in computation. Thus, degradation of response may occur due to disturbance, change in payload, and inaccurate sensor measurement. Their simulation showed the results of link mass estimation and of one joint position error tested on dynamics of only two-joint planar manipulator. More complete simulation results are required to show the robustness with respect to various desired paths of high velocity profiles which indeed affect the dynamics of the manipulator and the robustness in change or existence of payload.

[KO88] proposed decentralized control of robot manipulator using state and PI feedback. This paper seems to suggest that a static state feedback is indeed sufficient to stabilize the system about a constant setpoint(desired path), and system may still perform satisfactorily if the constant reference signal is replaced by a slowly time-varying signal. However, with the assumption of constant setpoint or slowly time-varying reference input, stability or robustness problems of the controller for the robot manipulator cannot be objectively proved for the general desired trajectories because it has skipped crucial problems from trajectory dependent dynamics of manipulator by simple assumption. One of the expected problems in applications of the proposed control methodology is that simulation studies were tested only for a planar two-link manipulator with simple desired setpoint.

[SL88] presented adaptive solutions based on the prior availability of an explicit, linearly parameterized representation of motion equations of the robot. To guarantee the global stability and asymptotically convergent error, sufficiently smooth desired trajectories are assumed. As a design strategy, they mathematically separated the plant motion equation into a nonlinear function assumed known and physical parameters unknown but constant. The nonlinear function part is compensated by the computed torque method, while unknown parameters is controlled by the PD controller. This algorithm requires the exact prior knowledge and linear factorization process in advance which is possible when the physical uncertainty is only on the mass property of the individual link.

[KH91] proposed an adaptive self-tuning controller for a robot. They improved the performance by taking into consideration the interactions between joints of the Stanford manipulator. This approach used the MIMO ARX model, and estimated (blocked) off-diagonal matrix to take into account interactions between joints. It is regarded as an almost black-box model control since they did not use pre-information of robot dynamics. This research, to a certain extent, improved convergence at moderate or high velocity after fine tuning the forgetting factor and weight constants for the optimal controller along given desired signals. But it has the drawback of a large tracking error when desired trajectories are changed to even slower velocity profiles. This method sometimes failed to converge to an arbitrary desired response of bounded-velocity profile. This defect is presumed from less robustness of the system identification carried out by the linear ARX model and classical estimation algorithm for the highly nonlinear time-varying dynamics. A more generalized system model and robust estimation algorithm are demanded to resolve drawbacks effectively.

[Spo92] devised the robust robot control law based on Lyapunov stability theory in conjunction with Lagrange-Euler robot dynamics equation. Their novel approach lies in the fact that uncertainty bounds for the control law are derived and the tracking error bounds is shown depending only on the inertia parameters of robot. In previous works of this type, the uncertainty bounds have depended not only on the inertia parameters but also on the desired trajectories and the manipulator state vector. However, this control strategy did not overcome the previous adaptive scheme by using the computed torque method requiring the exact pre-knowledge about the plant. Generally the computed torque method suffers from less capability for the uncertainty handling.

Other research of [LM91, LM93, ML93a, ML93b] used similar methods to Slotine and Li approaches as shown in [SL86, SL87a, SL87b, SL88, LS88] for the adaptive robot control. They made full use of pre-analysis of mathematical *linear factorization* on the rigid-body robot dynamics. This clone commonly suffers from limitation of an *ad-hoc* controller and uncertainty compensation problem.

1.3.3 Summary

Conventional nonlinear controls (generally non-ANN based adaptive controls) have similar limitations, which are also shown in computed torque control [LWP80],[LC85], Lyapunov based control [CL81],[Cor89], the sliding mode control [DZM88], [ND94], [UF94], and the exact linearization control [Tak93]. These limitations apply in general and more specifically to robot control. They include:

1. Control laws frequently use the computed torque method for robust control with exact knowledge about the system being controlled. This leads to an *ad-hoc* controller for a specific nonlinear system and requires heavy computation to evaluate the nonlinear terms analytically. Indirect adaptive control suffers from robustness and generalization strongly depending on the performance of the system identifier as shown in [KG83, SK87, KH91].

2. Indirect adaptive controllers have faced the difficulties of stability and robustness due to weak system identification capability depending on system models and estimation algorithms. This arises from its inherent design scheme such that it generally uses the inputs observation and outputs measurement only. A remedy for those difficulties is to use the ANN-based system identifier where the adequate design of neural nets and its effective training method are required.

3. To avoid the problems of stability and robustness, the operating range has been limited as an indirect solution as follows:

   (a) Some adaptive controllers have assumed slowly-moving speed, which may partly lead the original nonlinear control to a linear control problem and result in a decoupling effect of the original coupled system. This causes performance degradation or fails when the operating range is increased. Therefore the operating range should not be sacrificed.

   (b) Simulation scenarios need to perform on arbitrary trajectories for generalization. The simple planar two-link robot is not enough for a test-bed because the adaptive performance depends on the size of the system model.

4. Compensation ability of a nonlinear uncertainty needs to be tested through simulation works. This capability usually is realized through the robust system identification without using pre-knowledge. The importance of the system model is again emphasized in this point.

Since drawbacks and limitations are found in aforementioned non-neuromorphic adaptive controls, the use of neuromorphic control is getting widespread attention with some limitations listed below:

1. Some neural network architectures were built based on the complete available information about a system being controlled. This causes poor generalization properties and increases the structure complexity depending on the system. Therefore, the performance and success rate of this neural architecture are highly dependent on the tested system or test scenarios.
2. Feedforward (multi-layer) neural networks were frequently used for the dynamic mapping due to the difficulty of less systematic recurrent neural network architectures. The effective use of memory unit (time-delay) and systematic recurrent links was seldom found to make a more generalized recurrent neural network. Simple increase of the hidden-layer neurons cannot always be an effective way to improve the feedforward neural performance because training algorithm efficacy deteriorates with increased synaptic weights.
3. Most neural training algorithms are heavily depending on the BP algorithm[7] (or steepest-descent, or delta rule, or gradient search) although there exist drawbacks of slow convergence, sensitivity of the learning rate and momentum constant, weights initialization problem and local minimum. This is because the BP clone can train synaptic weights in the hidden-layer.
4. Other training methods of the hidden-layer are seldom identified without using the BP's error propagation backward because the clear teaching signal in the

---

[7] The BP algorithm was first documented in the Werbos's Ph.D. thesis [Wer74] at Harvard University in 1974.

hidden-layer is absent. To overcome the BP's bottleneck, a reasonable hidden-layer teacher should be developed without using the gradient search basis.

5. The Kalman filter family including the RLS algorithm was used for the single-layer training on-line due to its fast convergence speed. The intensive research of the numerical burst phenomenon, which is identified as a crucial drawback in the Kalman filter (RLS) clone, was frequently neglected in conjunction with the neural training method. The extension of the Kalman filter to the multi-layer training is desired, where however the derivation of a clear supervising signal for the hidden-layer training is necessary for better performance.

6. To cover less robustness of the on-line training algorithms, the undesirable uses of an off-line training method for a dynamic system were sometimes performed as a pre-processing step before the on-line training. This limits the flexibility of the on-line training algorithm.

1.4 Objectives Of the Thesis

The primary objective of this thesis was to develop a fast, accurate, robust, and generalized neural system identifier for on-line applications. This system identification technique has a wide variety of applications in different disciplines. One application of the on-line neural system identification scheme is the control of a nonlinear, time-varying and coupled robot system with uncertainties without using any pre-information about the plant. The designed control scheme is expected to be a universal nonlinear controller depending only on the measurement signals. The following procedure is used to realize the research objectives:

1. To design a more general recurrent neural network for application to various dynamic systems. A specific neural network design based on pre-information of a system is not flexible in applications, and gives its training algorithm less robustness and more computational load under different systems. This development includes:

- A more efficient neuron model can be devised.
- Meaningful neural synaptic links can be connected by time-delay units (memory) and by finding of new signal sources.
- New components in one-layer can be observed to create more data points.
- More systematic recurrent architecture is desired for the generalized improvement of performance.

2. To develop the non-gradient based training algorithm leading to a fast, accurate and robust training algorithm on-line. For this the following steps are presented:
    - A methodology is developed to apply the conventional RLS (recursive least-squares) algorithm for the multi-layer neural nets training to make use of its advantages after the pros and cons of the RLS algorithm are investigated.
    - The clear hidden-layer teaching signal is derived not using the error propagation backward.
    - The numerical burst phenomenon in the RLS algorithm should be identified and resolved.
    - The synaptic weights update law in the RLS algorithm can be re-derived to improve its robustness.
    - The variable forgetting factor in the RLS algorithm should be developed and tested.
    - The covariance matrix in the RLS algorithm should be kept from the divergence due to various input signals.
3. To simulate the developed neural system identifier to show its improved performance under reasonable scenarios. Any previous off-line training process should not be performed for this.
4. To devise the adaptive self-tuning controller with on-line neural system identifier for a robot manipulator tracking problem only depending on the measure-

ment signal. This type of controller should be robust with respect to dynamic uncertainty compensation and arbitrary time-varying desired signal.

- LOQ optimization technique is integrated into the neural system identifier on-line.
- All algorithms are tested on the closely-simulated robot manipulator to the 3-joint PUMA 600.

1.5 Thesis Layout

Chapter 1 introduces the motivation and methodology for this research. It provides literature reviews of related works of ANNs and neuromorphic controls. The detailed reviews are also discussed in each chapter depending on the matched subject.

Chapter 2 presents the robotics relating to the robot manipulator dynamics. Through this chapter, the insight of a nonlinear, time-varying coupled systems' characteristics can be identified along with their complexities. The derived closed form of the rigid robot dynamics is useful as a performance test by being able to simulate the motion of a nonlinear system. The generation of training data and test of controller on-line are carried out based on equations supported by this chapter.

Chapter 3 reviews the general background of neuron modeling, capabilities, application areas about ANNs discipline, which can provide a developer with an intuition to develop and/or improve through a *creative imitation* process.

Chapter 4 deals with development of the recurrent neural networks architecture called the SERNN (supervision and error recurrent neural networks) for a more robust and generalized system identification. Most architectural efforts are assigned to find a good recurrent neural net (i.e., a specific mathematical form through synaptic links) capable of producing output values that match desired values. In ANNs society, it is known that the recurrent architecture is superior to feedforward nets for the dynamic mapping capability. The conventional neural architecture of increasing simply the number of hidden-layer neurons is not desirable.

Chapter 5 develops a fast, accurate and robust on-line training method called the MRLS (modified recursive least-squares) algorithm. Most of ANNs training have been carried out by the error backpropagation clone. However, there are serious drawbacks of BP in a variety of research even though the BP algorithm can often find a good set of synaptic weights in a reasonable time. The fundamental defect in the BP clone is recognized as its use of the gradient descent optimization principle. The objective specification of this chapter is to develop the pure non-BP training algorithm. Even though there exist conventional estimation algorithms for a linear parametric model, their direct application to the multi-layer neural networks has been hampered because (of): (a) ANNs is a nonlinear model, (b) ANNs have a multi-layered topology leading to the absence of a teaching signal for the hidden-layers, (c) limitations of speed, accuracy and robustness in neural training process, and (d) performance sensitivity depending on initial weights values. Comprehensive trials are tackled to resolve these problems in this chapter.

Chapter 6 investigates how the developed neural architecture and training algorithms perform for the black-box neural system identification of dynamic nonlinear systems through simulations. This chapter shows excellent performances such as speed, accuracy, robustness and generalization for a neural identifier. A good system identification technology can be applied to a variety of adaptive processing areas. In this thesis, the neural identifier is used for the adaptive self-tuning control of a robot manipulator without using pre-information.

Chapter 7 presents how the on-line neural identifier is synthesized with the optimal LOQ (linearly observed quadratic) control processing leading to the adaptive self-tuning scheme. This control scheme is supposed to exhibit design specifications such as no restrictions about system linearity, robustness with respect to different variable trajectories, good capabilities for uncertainties including sudden dynamic changes in the middle of excursion, reasonable noise rejection ability, and no pre-information requirement about a nonlinear system being considered (only measured signals are used). It is expected that the pure measurement-dependent control (i.e., black-box model based control) would be useful for complex nonlinear systems which are difficult to be modeled by the mathematical analysis.

Chapter 8 concludes this thesis with contributions and further research.

Bibliography

[BGC96] L. Behera, M. Gopal, and S. Chaudhury. On adaptive trajectory tracking of a robot manipulator using inversion of its neural emulator. *IEEE Trans. on Neural Networks*, vol.7(no.6):pp.1401–1414, November 1996.

[Car91] J. Cardoso. Super-symmetric decomposition of the fourth-order cumulant tensor; blind identification of more sources than sensors. *Proc. IEEE ICASSP-91*, pages 3109–3112, 1991. Toronto.

[CBG90] S. Chen, S.A. Billings, and P.M. Grant. Non-linear system identification using neural networks. *Int. J. Control*, vol.51(no.6):pp.1191–1214, 1990.

[CCBG90] S. Chen, C.F.N. Cowan, S.A. Billings, and P.M. Grant. Parallel recursive error algorithm for training layered neural networks. *Int. J. Control*, vol.51(no.6):pp.1215–1228, 1990.

[CG89] H.-H. F. Chen and D.A. Guenther. Self-tuning optimal control of an active suspension. *SAE 892485, Society of Automotive Engineers*, pages 19–24, November 1989. Warrendale, PA 15096-0001.

[CHS86] J.J. Craig, P. Hsu, and S.Sastry. Adaptive control of mechanical manipulators. *IEEE Conference on Robotics and Automation, San Francisco*, April 1986.

[CJH91] P. Comon, C. Jutten, and J. Herault. Blind separation of sources, part ii: problem statement. *Signal Processing*, vol.24:pp.11–20, 1991.

[CL81] M. Corless and G. Leitmann. Continuous state feedback guaranteeing uniform ultimate boundedness for uncertain dynamic systems. *IEEE Trans. Automatic Control*, vol.26(no.5):pp.1139–1144, 1981.

[CM92] A. Cichocki and L. Moszczynski. New learning algorithm for blind separation of sources. *Electronics Letters*, vol.28(no.21):pp.1986–1987, 1992.

[Cor89] M. Corless. Tracking controllers for uncertain systems: Application to a manutec r3 robot. *Trans. ASME, J. of Dynamic Systems, Measurement, and Control*, vol.111(no.4):pp.609–618, 1989.

[CPA91] M.H. Cohen, P.O. Pouliquen, and A.G. Andreou. Silicon implementation of an auto-adaptive network for real-time separation of independent signals. *Proc. IEEE ISCAS-91*, pages 2971–2974, 1991. Singapore.

[CSB96] D.C. Chen, B.J. Sheu, and T.W. Berger. A compact neural network based on cdma receiver for multimedia wireless communication. *IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pages 99–103, 1996.

[Cyb89] G. Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, vol.2:pp.303–314, 1989.

[DD79] S. Dubowsky and D. DesForges. The application of model-referenced adaptive control to robotic manipulators. *ASME J. Dynam. Syst. Meas. Control 101,193*, 1979.

[DZM88] R.A. DeCarlo, S.H. Zak, and G.P. Mattews. Variable structure control of nonlinear multivariable systems: A tutorial. *Proc. of the IEEE*, vol.76(no.3):pp.212–232, 1988.

[HJ86] J. Herault and C. Jutten. Space or time adaptive signal processing processing by neural network models. *AIP Conf. Proc.*, pages 206–211, 1986. Snowbird, UT.

[IST92] Y. Iiguni, H. Sakai, and H. Tokumaru. A real time learning algorithm for a multilayered neural network based on extended kalman filter. *IEEE Trans. on Signal Processing*, vol.40(no.4), April 1992.

[JH91] C. Jutten and J. Herault. Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, vol.24:pp.1–10, 1991.

[JH95] Seul Jung and T.C. Hsia. On reference trajectory modification approach for cartesian space neural network control of robot manipulators. *IEEE Int. Conf. on Robotics and Automation*, pages 575–580, 1995.

[KFS87] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural network model for controlling and learning of voluntary movements. *Biological Cybernetics*, vol.57:pp.169–185, 1987.

[KG83] A.J. Koivo and T.H. Guo. Adaptive linear controller for robotic manipulators. *IEEE Transactions on Automatic Control*, AC-28(no.2):pp.162–171, Feb. 1983.

[KH91] A.J. Koivo and N. Houshangi. Real-time vision feedback for servoing robotic manipulator with self-tuning controller. *IEEE Transactions on Systems, Man and Cybernetics*, vol.21(no.21), Jan./Feb. 1991.

[KK85] P. Khosla and T. Kanade. Parameter identification of robot dynamics. *IEEE Conf. Decision Control, Fort Laudale, FL*, 1985.

[KM96] G.I. Kechriotis and E.S. Manolakos. Hopfield neural network implementation of the optimal cdma multiuser detector. *IEEE Trans. on Neural Networks*, vol.7(no.1):pp.131–141, January 1996.

[KO88] F. Khorrami and U. Ozguner. Decentralized control of robot manipulators via state and proportional-integral feedback. *Proc. of the 1988 IEEE International Conference on Robotics and automation*, Apr.24-29 1988. Philadelphia, PA, USA.

[KUIS88] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki. Hierarchical neural model for voluntary movement with application to robotics. *IEEE Control Systems Magazine*, vol.8(no.2):pp.8–16, 1988.

[KV92] D. Katic and M. Vukobratovic. Decomposed connectionist architecture for fast and robust learning of robot dynamics. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 2064–2069, May 1992. Nice.

[KV95] D.M. Katic and M.K. Vukobratovic. Highly efficient robot dynamics learning by decomposed connectionist feedforward control structure. *IEEE Trans. on Systems Man, and Cybernetics*, vol.25(no.1):pp.145–158, 1995.

[LC84] C.S.G. Lee and M.J. Chung. An adaptive control strategy for mechanical manipulators. *IEEE Trans. Automatic Control*, AC-29(no.9):pp.837–840, 1984.

[LC85] C. S. G. Lee and M.J. Chung. An adaptive perturbation control with feedforward compensation for robot manipulators. *Simulation*, vol.44(no.3):pp.127–136, 1985.

[LM91] W.-S. Lu and Q.-H. Meng. Recursive computation of manipulator regressor and its application to adaptive motion control of robots. *Proceedings 91 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, vol.1:pp.170–173, 1991.

[LM93] W.-S. Lu and Q.-H. Meng. Regressor formulation of robot dynamics: Computation and applications. *Trans. on Robotics and Applications*, vol.9(no.3):pp.323–333, June 1993.

[LP96] K.-N. Lou and R.A. Perez. A new system identification technique using kalman filtering and multilayer neural networks. *Artificial Intelligence Engineering*, vol.10:pp.1–8, 1996.

[LS88] W. Li and J.J.E. Slotine. Indirect adaptive robot control. *IEEE Int. Conf. Robotics and Automation*, vol.2:pp.704–709, 1988. Philadelphia, PA.

[LWP80] J.Y.S. Luh, M.W. Walker, and R.P.C. Paul. On-line computational scheme for mechanical manipulator. *Trans. ASME, J. of Dynamic Systems, Measurement, and Control*, vol.102(no.2):pp.69–76, 1980.

[MD93] M. McInerney and A.P. Dhawan. Use of genetic algorithms with back propagation in training of feed-forward neural networks. *IEEE Int. Conf. On Neural Networks*, vol.1:pp.203–208, 1993.

[ML93a] Q.-H. Meng and W.-S. Lu. An adaptive control scheme for robot manipulators with flexible joints. *Proc. of the 3th Midwest Symposium on Circuits and Systems*, vol.1:pp.394–396, 1993.

[ML93b] Q.-H. Meng and W.-S. Lu. A neural network adaptive control scheme for robot manipulator. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, vol.2:pp.606–609, 1993.

[ND94] K. Nonami and K. Den. *Sliding Mode Control.* Corona Publishing Co., Ltd, 1994.

[OS89] R. Ortega and M.M. Spong. Adaptive motion control of rigid robots: a tutorial. *Automatica*, vol.25(no.6):pp.877–888, 1989.

[OSF$^+$91] T. Ozaki, T. Suzuki, T. Furuhashi, S. Okuma, and Y. Uchikawa. Trajectory control of robotic manipulators. *IEEE Trans. on Systems, Man and Cybernetics*, vol.1(no.1):pp.273–280, 1991.

[SK87] R. Souissi and A.J. Koivo. Design of adaptive self-tuning controller for high speed manipulators. *Proc. of the 1987 IEEE International Conference on Systems, Man and Cybernetics*, vol.3, Oct.20-23 1987. Alexandria, Virginia.

[SL86] J.J.E. Slotine and W. Li. Adaptive manipulator control. *A.S.M.E. Winter Annual Meeting, Anaheim, CA*, 1986.

[SL87a] J.J.E. Slotine and W. Li. Adaptive strategies in constrained manipulation. *IEEE Int. Conf. Robotics and Automation*, vol.2:pp.595–601, 1987. Raleigh,NC.

[SL87b] J.J.E. Slotine and W. Li. On the adaptive control of robot manipulators. *IEEE Int. J. Robotics Research*, vol.3(no.3):pp.49–59, 1987.

[SL88] J.J.E. Slotine and W. Li. Adaptive manipulator control: A case study. *IEEE Trans. Autom. Control*, vol.33(no.11):pp.995–1003, 1988.

[SMS91] D.S. Shook, C. Mohtadi, and S.L. Shah. Identification for long-range predictive control. *IEE Proc. D Control and Applications*, vol.138, Iss1:pp.75–84, Jan. 1991.

[SO90] M.M. Spong and R. Ortega. On adaptive inverse dynamics control of rigid robots. *IEEE Trans. on Automatic Control*, vol.35(no.1):pp.92–95, Jan. 1990.

[Sor91] E. Sorouchyari. Blind separation of sources, part iii: stability analysis. *Signal Processing*, vol.24:pp.21–29, 1991.

[Spo92] Mark M. Spong. On the robust control of robot manipulators. *IEEE Trans. on Automatic Control*, vol.37(no.11):pp.1782–86, 1992.

[SS92] R.M. Sanner and J.J.E. Slotine. Gaussian networks for direct adaptive control. *IEEE Trans. Neural networks*, vol.3(no.6):pp.837–863, 1992.

[SS95] R.M. Sanner and J.J.E. Slotine. Stable adaptive control of robot manipulators using neural networks. *Neural Computation*, vol.7:pp.753–790, 1995.

[ST92] R.S. Scalero and N. Tepedelenlioglu. A fast new algorithm for training feedforward neural networks. *IEEE Trans. on Signal Processing*, vol.40(no.1):pp.202–1992, January 1992.

[SV89] Mark W. Spong and M. Vidyasagar. *Robot dynamics and control.* John Wiley & Sons, Inc., 1989.

[TA81] M. Takegaki and S. Arimoto. A new feedback method for dynamic control of manipulators. *Trans. ASME, J. Dynamic Systems, Measurement and Control*, vol.102:pp.119–125, 1981.

[Tak93] Y. Takagi. A modeling and control system design theory for nonlinear systems. *Systems, Control and Information*, vol.37(no.1):pp.23–29, 1993.

[TLSH91] L. Tong, R. Liu, V.-C. Soon, and Y.-F. Huang. Indeterminacy and identifiability of blind identification. *IEEE Trans. on Circuit and Systems*, vol.38:pp.499–509, 1991.

[TS96] W.G. Teich and Michael Seidl. Code division multiple access communications: multiuser detection based on a recurrent neural networks structure. *IEEE Int. Symposium on Spread Spectrum Techniques and Applications*, vol.3:pp.979–985, 1996.

[UF94] V.I. Utkin and A. Ferrara. Special issues on sliding mode control. In A. Ferrara, editor, *Control Theory and Advanced Technology*. Mita Press, 1994.

[Vac91] R. Vaccaro. *SVD and Signal Processing: Algorithms and Architecture*. North-Holland, Amsterdam, 1991.

[VM88] J. Vandewalle and B. De Moore. A variety of applications of singular value decomposition in identification and signal processing. In E.F. Deprettere, editor, *SVD and Signal Processing*, pages 43–91, Amsterdam, 1988. Elsevier.

[VSK85] M. Vukobratovic, D. Stokic, and N. Kircanski. *Non-adaptive and adaptive control of manipulation robots.* Spinger Verlag, Berlin, 1985.

[Wer74] P.J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, Cambridge, MA, August 1974.

[YB89] D.Y. Yeung and G.A. Beckey. Using a context-sensitive learning network for robot arm control. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages pp.1441–1447, May 1989. Scottsdale.

Chapter 2

Robotics For Computer Simulation

This chapter introduces the robotic kinematics and the derivation of the dynamics on the PUMA 600. The robotic information presented here is useful for digitally simulating the behavior of the robot arm. The input and output measurements of the plant are necessary for the system identification purpose and sometimes for the evaluation of the controller performance. General discussion of the robot systems is presented, and the techniques are described in conjunction with PUMA 600 manipulator.

2.1 Description of Manipulator by D-H Notation

The research [4] first proposed a systematic method to define a local coordinate system for each link of a robot manipulator. The relations between a world coordinate system and a local coordinate system will be described by link parameters and joint variables from Denavit-Hartenberg (D-H) coordinate system(frame), which are often called *kinematic parameters.* A schematic picture of PUMA 600 manipulator is shown in Figure 4 where the D-H coordinate system and its kinematic parameters are used.

2.1.1 Determination of Coordinate System in Robotics

The PUMA 600 series robot manipulator consists of seven links connected by six revoluted joints. Since the motion of the links involves angular rotation, the position

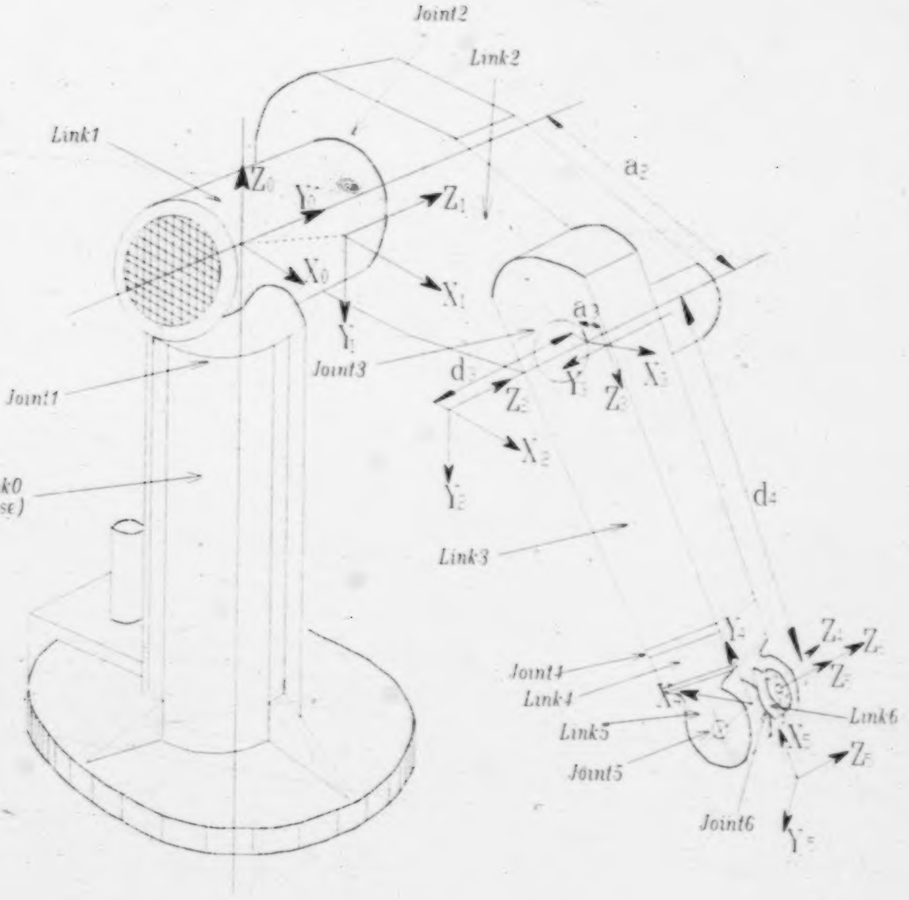

Joint2
Link2
Link1
a2
Z0
Y0
Z1
X0
X1
Y1
Joint1
Joint3
a3
d3
Z2
Y3
Z3
X3
X2
Y2
Link0
(Base)
d4
Link3
Y4
Z4
Z6
Z5
Joint4
Link4
X4
Link5
Link6
X5
Z5
Joint5
Joint6
Y5

Figure 4: PUMA Manipulator with D-H Notation

and orientation of the end-effector has definite relationships with joint angles (or joint variables) and links. These mathematical relations depend on the coordinate systems chosen, and play an important role in the kinematics, dynamics and the control of robot manipulator. Since the PUMA 600 has six joints and seven links, we can assign one coordinate system to each link. One coordinate system whose origin can be fixed to the base, or at the shoulder of the manipulator, is called the *World Coordinate System*, while the six coordinate systems whose origins can be assigned to the each of the six links are called *Local Coordinate Systems*. Accordingly the local coordinate system(frame) moves with the links.

The problem is to assign rectangular right-hand coordinate frames and kinematic parameters for the links. Then, the transformation matrices relating to the coordinate frames are to be written. For the next link $i+1$ with respect to link i in Figure 4, the coordinate axes are chosen by the D-H coordinate frame assigning method [4, 1, 7].

2.1.2 Determination of the Kinematic Parameters

After establishing the coordinate frames, it is necessary to define four kinematic parameters which are sometimes called the *structural kinematic parameters* since they depend on the structure of the given manipulator and describe the relative position of a successive pair of the axes in two coordinate systems. Having established four structural parameters d_i, a_i, α_i, and θ_i, $i = 1, \cdots, n$ for a particular manipulator, the transformation matrix between the adjacent coordinate frames may be written.

For the purpose of illustrating four structural parameters, a coordinate system shown in Figure 5 can be described in terms of four parameters - *length* a_1, the *twist angle* α_1, *distance* d_1, and *angle* θ_1 between links.

In Figure 5, since a straight line L is assumed to represent a rotational axis, a local coordinate frame is to be assigned. For a multiple joint serial link manipulator, the aforementioned structural parameters are determined for the ith link, $i = 1, \cdots, n$ as follows:

1. a_i : The distance from the origin of the ith coordinate frame to the intersection

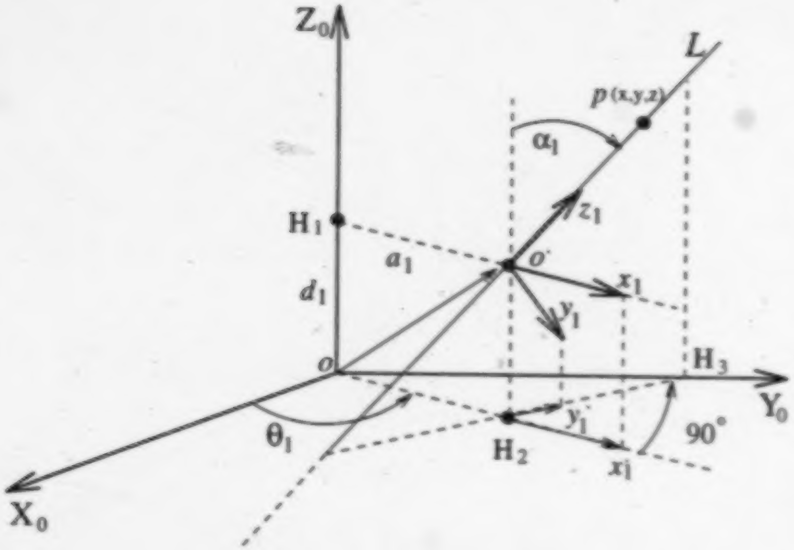


Figure 5: Illustration of Structural Parameters

of the Z_{i-1}- and the X_i-axis along the x_i-axis.

2. α_i : The angle of rotation about the positive X_i-axis is measured from the positive Z_{i-1} (or its parallel projection) to the positive Z_i-axis, where the positive direction is counterclockwise.

3. θ_i : The angle of rotation about the positive Z_{i-1} is measured from the positive X_{i-1}-axis to the positive X_i-axis. It is positive in the counterclockwise direction.

4. d_i : The distance from the origin of the $(i-1)$st coordinate frame to the intersection of the Z_{i-1}-axis, and the X_i-axis along the Z_{i-1}-axis.

Based on the procedures to determine the structural kinematic parameters, the values of d_i, a_i, α_i and θ_i [13] can be obtained for PUMA 600 in Table 2.1 through Figure 4.

joint $i =$	α_i°	θ_i°	d_i	a_i
1	-90	θ_1	0	0
2	0	θ_2	0	a_2
3	90	θ_3	d_3	a_3
4	-90	θ_4	d_4	0
5	90	θ_5	0	0
6	0	θ_6	0	0

Table 2.1: Structural Kinematic Parameters

2.2 Determination of Transformation Matrix

A 4 × 4 matrix, which is called the *transformation matrix*, relates the link-attached coordinate frame to the reference coordinate frame.

2.2.1 Recursive Transformation Matrix

If vector P_i is known in the ith coordinate frame, then it can be expressed in the $(i-1)$st coordinate frame as P_{i-1}, that is

$$P_{i-1} = A_{i-1}^{i} P_i \tag{2.1}$$

where matrix A_{i-1}^{i} of the *recursive transformation matrix* can be written by the following general form:

$$A_{i-1}^{i} = \left[\begin{array}{ccc|c} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.2}$$

We can extend equation 2.1 as follows:

$$\begin{aligned} P_0 &= A_0^1 A_1^2 A_2^3 A_3^4 A_4^5 A_5^6 \, P_6 \\ &= A_0^6 \, P_6 \end{aligned} \tag{2.3}$$

Matrix A_0^6 in Equation 2.3 is a function of structural kinematic parameters. When these values are known, the position of the end-effector can be calculated by Equation 2.3. The determination of this position from the values of the joint variables is referred to as solving the ***forward kinematic equations***.

2.2.2 Partition and Decomposition of Transformation Matrix

Transformation matrix A_0^6 can be *partitioned* into four parts as follows:

$$\begin{aligned} A_0^6 &= \left[\begin{array}{c|c} \text{Rotation Submatrix} & \text{Translation Vector} \\ (3 \times 3) & (3 \times 1) \\ \hline \text{Perspective Vector} & \text{Scaling Factor} \\ (1 \times 3) & (1 \times 1) \end{array}\right] \\ &= \left[\begin{array}{ccc|c} X_x & Y_x & Z_x & P_x \\ X_y & Y_y & Z_y & P_y \\ X_z & Y_z & Z_z & P_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \end{aligned} \tag{2.4}$$

And also the transformation matrix A_0^6 can be ***decomposed*** into matrix multiplication of translation matrix-$Trans(P_x, P_y, P_z)$ and rotation matrix-$RPY(\Psi_x, \Psi_y, \Psi_z)$ as follows:

$$\begin{aligned} A_0^6 &= Trans(P_x, P_y, P_z) RPY(\Psi_x, \Psi_y, \Psi_z) \\ &= \left[\begin{array}{ccc|c} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc|c} X_x & Y_x & Z_x & 0 \\ X_y & Y_y & Z_y & 0 \\ X_z & Y_z & Z_z & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \end{aligned} \tag{2.5}$$

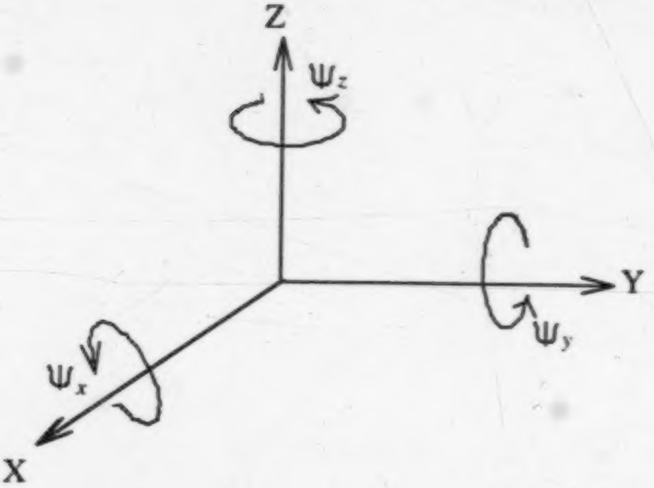


Figure 6: R-P-Y Angles for Orientation

2.3 Orientation of End-Effector

Grasping of an object by the end-effector means that the position and orientation of the end-effector are the same as those of the object in the reference coordinate frame. The simplest one to understand is roll(Ψ_z), pitch(Ψ_y), and yaw(Ψ_x) in Figure 6 which are called *R-P-Y angles*. These R-P-Y angles are used to represent the overall orientation of the object or end-effector of the manipulator.

A sequence of three rotations is a rotation about the X axis, a rotation about the Y axis, and a rotation about the Z axis. So the rotation matrix, $RPY(\Psi_x, \Psi_y, \Psi_z)$, can be performed by means of successive rotational operations:

$$RPY(\Psi_z, \Psi_y, \Psi_x) = Rot(z, \Psi_z)Rot(y, \Psi_y)Rot(x, \Psi_x) =$$

$$\left[\begin{array}{ccc|c}
c(\Psi_z)c(\Psi_y) & c(\Psi_z)s(\Psi_y)s(\Psi_x) - s(\Psi_z)c(\Psi_x) & c(\Psi_z)s(\Psi_y)c(\Psi_x) + s(\Psi_z)s(\Psi_x) & 0 \\
s(\Psi_z)c(\Psi_y) & s(\Psi_z)s(\Psi_y)s(\psi_x) + c(\Psi_z)c(\Psi_x) & s(\Psi_z)s(\Psi_y)c(\Psi_x) - c(\Psi_z)s(\Psi_x) & 0 \\
-s(\Psi_y) & c(\Psi_y)s(\Psi_x) & c(\Psi_y)c(\Psi_x) & 0 \\
\hline
0 & 0 & 0 & 1
\end{array}\right]$$

(2.6)

where

$$Rot(z,\Psi_z) = \begin{bmatrix} c(\Psi_z) & -s(\Psi_z) & 0 & 0 \\ s(\Psi_z) & c(\Psi_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(y,\Psi_y) = \begin{bmatrix} c(\Psi_y) & 0 & s(\Psi_y) & 0 \\ 0 & 1 & 0 & 0 \\ -s(\Psi_y) & 0 & c(\Psi_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(x,\Psi_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\Psi_x) & -s(\Psi_x) & 0 \\ 0 & s(\Psi_x) & c(\Psi_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $s(\cdot)$ and $c(\cdot)$ stand for $\sin(\cdot)$ and $\cos(\cdot)$ respectively. The $Rot(\cdot)$ matrices are called the ***basic homogeneous rotation matrices.***

2.3.1 An Algorithm for Calculating Ψ_x, Ψ_y, and Ψ_z

R-P-Y angles for the orientation of the manipulator need to be computed with the given four structural parameters. The transformation matrix A_0^6 for the PUMA robot manipulator can be calculated by successive multiplication of the transformation matrix in equation 2.2 with the structural kinematic parameters in Table 2.1. This successive matrix multiplication can be easily performed by using symbolic computation software, for example, *Maple V* developed by the University of Waterloo [15]. This result is equated by equation 2.4 for calculating $Roll(\Psi_z)$,$Pitch(\Psi_y)$, and $Yaw(\Psi_x)$ angles, i.e.,

$$\left.\begin{aligned}
X_x &= c_6c_5c_4c_1c_2c_3 - c_6c_5c_4c_1s_2s_3 - c_6c_5s_1s_4 - c_6s_5c_1c_2s_3 - c_6s_5c_1s_2c_3 \\
&\quad -s_6s_4c_1c_2c_3 + s_6s_4c_1s_2s_3 - s_6s_1c_4 \\
Y_x &= -s_6c_5 - c_4c_1c_2c_3 + s_6c_5c_4c_1s_2s_3 + s_6c_5s_1s_4 + s_6s_5c_1c_2s_3 \\
&\quad +s_6s_5c_1s_2c_3 - c_6s_4c_1c_2c_3 + c_6s_4c_1s_2s_3 - c_6s_1c_4 \\
Z_x &= s_5c_4c_1c_2c_3 - s_5c_4c_1s_2s_3 - s_5s_1s_4 + c_5c_1c_2s_3 + c_5c_1s_2c_3 \\
P_x &= d_4c_1c_2s_3 + d_4c_1s_2c_3 + c_1c_2a_3c_3 - c_1s_2a_3s_3 - s_1d_3 + c_1a_2c_2 \\[1ex]
X_y &= c_6c_5c_4s_1c_2c_3 - c_6c_5c_4s_1s_2s_3 + c_6c_5c_1s_4 - c_6s_5s_1c_2s_3 - c_6s_5s_1s_2c_3 \\
&\quad s_6s_4s_1c_2c_3 + s_6s_4s_1s_2s_3 + s_6c_1c_4 \\
Y_y &= -s_6c_5c_4s_1c_2c_3 + s_6c_5c_4s_1s_2s_3 - s_6c_5c_1s_4 + s_6s_5s_1c_2s_3 \\
&\quad -c_6s_4s_1c_2c_3 + c_6s_4s_1s_2s_3 + c_6c_1c_4 + s_6s_5s_1s_2c_3 \\
Z_y &= s_5c_4s_1c_2c_3 - s_5c_4s_1s_2s_3 + s_5c_1s_4 + c_5s_1c_2s_3 + c_5s_1s_2c_3 \\
P_y &= d_4s_1c_2s_3 + d_4s_1s_2c_3 + s_1c_2a_3c_3 - s_1s_2a_3s_3 + c_1d_3 + s_1a_2c_2 \\[1ex]
X_z &= -c_6c_4c_5s_2c_3 - c_6c_4c_5c_2s_3 + c_6s_5s_2s_3 - c_6s_5c_2c_3 + s_4s_6s_2c_3 \\
&\quad +s_4s_6c_2c_3 \\
Y_z &= s_6c_4c_5s_2c_3 + s_6c_4c_5c_2s_3 - s_6s_5s_2s_3 + s_6s_5c_2c_3 + s_4c_6s_2c_3 \\
&\quad +s_4c_6c_2s_3 \\
Z_z &= -c_4s_5s_2c_3 - c_4s_5c_2s_3 - c_5s_2s_3 + c_5c_2c_3 \\
P_z &= -d_4s_2s_3 + d_4c_2c_3 - s_2a_3c_3 - c_2a_3s_3 - a_2s_2.
\end{aligned}\right\}$$

(2.7)

But transformation matrix A_0^6 can be rewritten in terms of *Translation Matrix* and basic *Rotation Matrices* about $X - Y - Z$ axes:

(2.8) $$A_0^6 = Trans(P_x, P_y, P_z)Rot(z, \Psi_z)Rot(y, \Psi_y)Rot(x, \Psi_x)$$

By rearranging equation 2.8:

(2.9) $$Rot^{-1}(z, \Psi_z)Trans^{-1}(P_x, P_y, P_z)A_0^6 = Rot(y, \Psi_y)Rot(x, \Psi_x)$$

The left hand side of equation 2.9 is:

$$Rot^{-1}(z, \Psi_z)Trans^{-1}(P_x, P_y, P_z)A_0^6$$

$$= \left[\begin{array}{ccc|c} c(\Psi_z) & s(\Psi_z) & 0 & 0 \\ -s(\Psi_z) & c(\Psi_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc|c} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc|c} X_x & Y_x & Z_x & P_x \\ X_y & Y_y & Z_y & P_y \\ X_z & Y_Z & Z_z & P_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} X_x c(\Psi_z) + X_y s(\Psi_z) & Y_x c(\Psi_z) + Y_y s(\Psi_z) & Z_x c(\Psi_z) + Z_y s(\Psi_z) & 0 \\ X_y c(\Psi_z) - X_x s(\Psi_z) & Y_y c(\Psi_z) - Y_x s(\Psi_z) & Z_y c(\Psi_z) - Z_x s(\Psi_z) & 0 \\ X_z & Y_z & Z_z & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(2.10)

The right hand side of equation 2.9 is:

$$Rot(y, \Psi_y)Rot(x, \Psi_x)$$

(2.11)

$$= \left[\begin{array}{ccc|c} c(\Psi_y) & 0 & s(\Psi_y) & 0 \\ 0 & 1 & 0 & 0 \\ -s(\Psi_y) & 0 & c(\Psi_y) & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & c(\Psi_x) & -s(\Psi_x) & 0 \\ 0 & s(\Psi_x) & c(\Psi_x) & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} c(\Psi_y) & s(\Psi_y)s(\psi_x) & s(\Psi_y)c(\Psi_x) & 0 \\ 0 & c(\Psi_x) & -s(\Psi_x) & 0 \\ -s(\Psi_y) & c(\Psi_y)s(\Psi_x) & c(\Psi_y)c(\Psi_x) & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

By comparing equation 2.10 and equation 2.11, we can calculate Ψ_z, Ψ_y, and Ψ_x as follows:

$\mapsto X_y c(\Psi_z) - X_x s(\Psi_z) = 0$

(2.12)

$$\longrightarrow \underline{\Psi_z = atan2(X_y, X_x)}$$

$$\mapsto X_x \cos(\Psi_z) + X_y \sin(\Psi_z) = \cos(\Psi_y), \; X_z = -\sin(\Psi_y)$$

$$\tan(\Psi_y) = \frac{\sin(\Psi_y)}{\cos(\Psi_y)} = \frac{-X_z}{X_x \cos(\Psi_z) + X_y \sin(\Psi_z)}$$

$$\text{(2.13)} \qquad \longrightarrow \underline{\Psi_y = atan2(-X_z, X_x \cos(\Psi_z) + X_y \sin(\Psi_z)}$$

$$\mapsto Y_z = \cos(\Psi_y)\sin(\Psi_x), \; Z_z = \cos(\Psi_y)\cos(\Psi_x)$$

$$\tan(\Psi_x) = \frac{\sin(\Psi_x)}{\cos(\Psi_x)} = \frac{Y_z}{Z_z}$$

$$\text{(2.14)} \qquad \longrightarrow \underline{\Psi_x = atan2(Y_z, Z_z)}$$

where argument values of $atan2()$ in equations 2.12 - 2.14 are given in equation 2.7. $atan2(y, x)$ returns the arctangent of y/x, in the range of $-\pi$ to π, using the signs of both arguments to determine the quadrant of the return value.

2.4 Summary - Need for Six-joint Dynamics

The position of the end-effector origin, i.e., P_x, P_y and P_z in equation 2.7 is only a function of the first three joint variables, θ_1, θ_2 and θ_3 instead of all six joint variables, $\theta_1, \cdots, \theta_6$, because other structural kinematic parameters are constants in Table 2.1. On the other hand, R-P-Y angles, Ψ_x, Ψ_y, and Ψ_z in equations 2.12 - 2.14, which are representing the orientation of the end-effector, depend on all six joint-variables.

2.5 Mathematical Modeling of PUMA Manipulator Dynamics

The dynamics equations or dynamics models describe the motion of a manipulator by means of differential or difference equations (or partial differential equations for a robot manipulator with nonrigid links). The equations of motion are useful for computer simulation and the design of a controller for a robot manipulator. In this section, we shall present the dynamics model based on *Lagrange-Euler* equation and its customized closed-form for the PUMA 600.

The derivation of the dynamics model of a robot manipulator based on the Lagrange-Euler equation is systematic [17, 1, 12]. By assuming rigid body motion, the resulting model of motion is a set of second-order coupled nonlinear differential equations. Using the 4×4 transformation matrix representation of the kinematic chain and the Lagrangian equation, [1] has shown that the dynamic motion equations for a six degree-of-freedom Stanford robot manipulator are highly nonlinear and consist of inertia loading, coupling reaction forces between joints (i.e., Coriolis and centrifugal) and gravity loading effects. These torques/forces depend on the robot manipulator's physical parameters, instantaneous joint position, velocity and acceleration, and the load it is carrying.

The L-E dynamics model of motion provides explicit state equations for the manipulator dynamics and can be utilized to analyze and design advanced control strategies in joint-variable space. This L-E dynamics model is used to solve the *forward dynamics* problem; that is, given the desired torques/forces, the equations of the dynamics model are used to obtain the joint accelerations which are then integrated to solve for joint positions and their velocities; or for the *inverse dynamics* problem, that is, given the desired joint positions, velocities and accelerations, generalized forces/torque are computed.

Unfortunately, the computation of these coefficients, especially in the case of a six-joint (d-o-f) robot manipulator, requires a fair amount of arithmetic operations even though its formulations are straightforward, systematic and readable. Thus, the L-E model is very difficult to utilize directly for real-time control purposes unless they are simplified. So, complete closed-form equations of models are seldom presented for a more than three d-o-f manipulator although the literature abounds with formulations for generating complete dynamics robot models [10, 3].

However, [11] has performed a complete and customized closed-form dynamics model of the six d-o-f PUMA robot manipulator in real-time using the symbolically processing software - ARM (Algebraic Robot Modeler), where the unmodeled dynamics like friction, backlash, and dynamics of actuator exist.

[11]'s customized closed-form of the dynamics model, which is based on L-E equation and customized with some parameters from [13], and which has reduced the computational requirements, is useful for simulating the six d-o-f PUMA robot manipulator on a computer and precompensating the nonlinear terms in the design of an adaptive controller. The customized closed-form equations of dynamics model for PUMA 600 robot manipulator, written in MATLAB, are presented in Appendix A.

2.6 Derivation of L-E Dynamics Model for Motion of Robot Manipulator

For control analysis and design, researchers would like to obtain an explicit set of closed-form differential equations (or state equations) that describe the dynamic behavior of a robot manipulator. In addition, the interaction and coupling reaction forces in the equations should be easily identified so that a proper controller can be designed to compensate for their effects. The Lagrange-Euler Dynamics model is one of the good models for the dynamics analysis [5].

2.6.1 Lagrange-Euler Equation

To determine a differential equation model for the motion of n d-o-f robot manipulator, the Lagrangian dynamics technique is employed. For a *conservative* system, the *Lagrange-Euler equation* is shown as below:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = 0 \qquad i = 1, \ldots, n \tag{2.15}$$

When external forces are acting on the system, they are included on the right hand side of equation 2.15. Thus, Lagrange-Euler equation for a *nonconservative* system is represented by the following form:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \qquad i = 1, \ldots, n \tag{2.16}$$

where

$\mathcal{L}$ = (Lagrange function) = kinetic energy(K) − potential energy(P)

K = total kinetic energy of the robot manipulator

P = total potential energy of the robot manipulator

q_i = generalized coordinates of the robot manipulator

$\dot{q}_i$ = first time derivative of the generalized coordinate, q_i

τ_i = generalized force(or torque) applied to the system
at joint i to drive link i

In order to determine the Lagrange-Euler(L-E) dynamics model from the Lagrange-Euler equation for a nonconservative system, one is required to properly choose a set of generalized coordinates. For example, $q_i \equiv \theta_i$ corresponds to generalized coordinates of joint variables which are defined in each of the 4 × 4 transformation matrices shown in equation 2.2.

The following derivation of the L-E dynamics model of a n d-o-f robot manipulator is based on transformation matrices in Chapter 2.

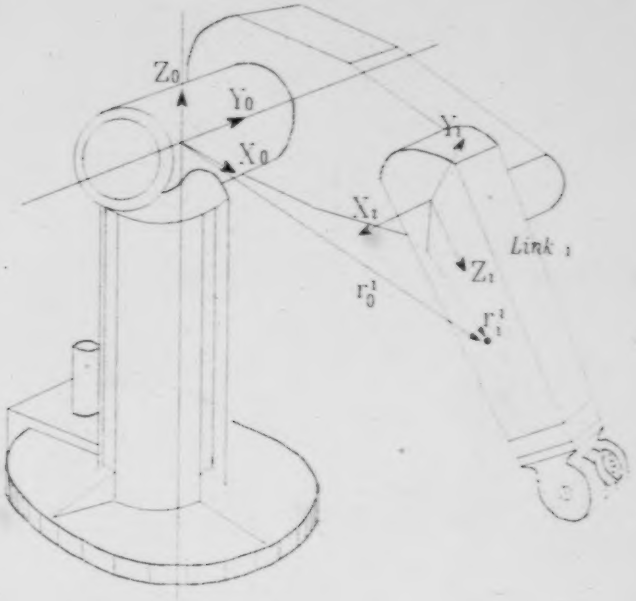


Figure 7: A Point $\mathbf{r}_i^i$ in Link i.

2.6.2 Kinetic Energy of a Robot Manipulator

The Lagrange-Euler equation 2.16 requires knowledge of the kinetic energy (K) of the physical system, which in turn requires knowledge of the velocity of each joint. In this section, the velocity of a point fixed in link i will first be derived and the effects of the motion of other joints on all points in this link will be explored.

As shown in Figure 7, let $\mathbf{r}_i^i$ be a point in the link i, and $\mathbf{r}_0^i$ be the same point as $\mathbf{r}_i^i$ with respect to the base coordinate frame. The transformation matrix A_{i-1}^i relates the spatial displacement of the ith coordinate frame to the $(i-1)$th link coordinate frame as follows:

$$\mathbf{r}_i^i = \begin{bmatrix} x_i^i \\ y_i^i \\ z_i^i \\ 1 \end{bmatrix} = [x_i^i,\ y_i^i,\ z_i^i,\ 1]' \tag{2.17}$$

$$\mathbf{r}_0^i = \begin{bmatrix} x_0^i \\ y_0^i \\ z_0^i \\ 1 \end{bmatrix} = [x_0^i,\ y_0^i,\ z_0^i,\ 1]' \tag{2.18}$$

The velocity of $\mathbf{r}_0^i$ then is:

$$\dot{\mathbf{r}}_0^i = \frac{d}{dt}\left(\mathbf{r}_0^i\right) = \begin{bmatrix} \dot{x}_0^i \\ \dot{y}_0^i \\ \dot{z}_0^i \\ 0 \end{bmatrix} \tag{2.19}$$

$$\mathbf{r}_0^i = A_0^i r_i^i \tag{2.20}$$

where

$$A_0^i = A_0^1 A_1^2 \cdots A_{i-1}^i$$

The velocity of $\mathbf{r}_i^i$ expressed in the base coordinate frame can be expressed as:

$$\begin{aligned}
\dot{\mathbf{r}}_0^i &= \frac{d}{dt}(\mathbf{r}_0^i) \\
&= \frac{d}{dt}(A_0^i \mathbf{r}_i^i) \\
&= \frac{d}{dt}\left(A_0^1 A_1^2 \cdots A_{i-1}^i \mathbf{r}_i^i\right) \\
&= \frac{d}{dt}\left(\dot{A}_0^1 \cdots A_{i-1}^i \mathbf{r}_i^i + A_0^1 \dot{A}_1^2 \cdots \mathbf{r}_i^i + \cdots + A_0^1 \cdots \dot{A}_{i-1}^i \mathbf{r}_i^i + A_0^1 \cdots A_{i-1}^i \dot{\mathbf{r}}_i^i\right) \\
&= \left(\frac{\partial A_0^i}{\partial q_1}\dot{q}_1 + \frac{\partial A_0^i}{\partial q_2}\dot{q}_2 + \cdots + \frac{\partial A_0^i}{\partial q_i}\dot{q}_i\right)\mathbf{r}_i^i \\
&= \left(\sum_{j=1}^{i} \frac{\partial A_0^i}{\partial q_j}\dot{q}_j\right)\mathbf{r}_i^i \quad , \quad \text{where} \quad \dot{\mathbf{r}}_i^i = 0 \qquad (2.21)
\end{aligned}$$

After obtaining a point velocity of each link i, we need to find the kinetic energy of link i. Let K_0^i be the kinetic energy of link i expressed in the base coordinate frame (which is an inertial frame), and dK_0^i be the kinetic energy of a particle with differential mass dm_i in link i, then the kinetic energy of the differential mass dm_i is:

$$\begin{aligned} dK_0^i &= \frac{1}{2}\left((\dot{x}_0^i)^2 + (\dot{y}_0^i)^2 + (\dot{z}_0^i)^2\right) dm_i \\ &= \frac{1}{2}\left(\dot{\mathbf{r}}_0^i \cdot \dot{\mathbf{r}}_0^i\right) dm_i \\ &= \frac{1}{2} Tr\left[\dot{\mathbf{r}}_0^i (\dot{\mathbf{r}}_0^i)'\right] dm_i \\ &= \frac{1}{2} Tr\left[\sum_{j=1}^{i} \frac{\partial A_0^i}{\partial q_j}\dot{q}_j \mathbf{r}_i^i \left(\sum_{k=1}^{i} \frac{\partial A_0^i}{\partial q_k}\dot{q}_k \mathbf{r}_i^i\right)\right] dm_i \\ &= \frac{1}{2} Tr\left[\sum_{j=1}^{i}\sum_{k=1}^{i} \frac{\partial A_0^i}{\partial q_j}\mathbf{r}_i^i \mathbf{r}_i^{i\prime} \frac{\partial A_0^{i\prime}}{\partial q_k}\dot{q}_j\dot{q}_k\right] dm_i \qquad (2.22) \end{aligned}$$

where $Tr[\cdot]$, a trace operator instead of a vector dot product, is used in the above equation. Then, the kinetic energy of link i is:

$$\begin{aligned} K_0^i &= \int dK_0^i \\ &= \frac{1}{2} Tr\left[\sum_{j=1}^{i}\sum_{k=1}^{i} \frac{\partial A_0^i}{\partial q_j}\left(\int \mathbf{r}_i^i \mathbf{r}_i^{i\prime} dm_i\right) \frac{\partial A_0^{i\prime}}{\partial q_k}\dot{q}_j\dot{q}_k\right] \\ &= \frac{1}{2} Tr\left[\sum_{j=1}^{i}\sum_{k=1}^{i} \frac{\partial A_0^i}{\partial q_j}(J_i) \frac{\partial A_0^{i\prime}}{\partial q_k}\dot{q}_j\dot{q}_k\right] \qquad (2.23) \end{aligned}$$

In equation 2.23, the integral can be put inside bracket since $\frac{\partial A_0^i}{\partial q_j}$ (that is, the rate of change of the point($\mathbf{r}_i^i$) in link i relative to base coordinate frame as q_j changes) is constant for all points on link i and independent of the mass distribution of the link, and also $\dot{q}_i$ are independent of the mass distribution of link i. The integral term inside parenthesis in equation 2.23 known as *pseudo-inertia matrix*, J_i is represented by recalling $\mathbf{r}_i^i = [x_i^i \;\; y_i^i \;\; z_i^i \;\; 1]'$ as follows:

$$
\begin{aligned}
J_i &= \int \mathbf{r}_i^i \mathbf{r}_i^{i\prime} dm_i \\
&= \begin{bmatrix} \int x_i^{i2} dm_i & \int x_i^i y_i^i dm_i & \int x_i^i z_i^i dm_i & \int x_i^i dm_i \\ \int x_i^i y_i^i dm_i & \int y_i^{i2} dm_i & \int y_i^i z_i^i dm_i & \int y_i^i dm_i \\ \int x_i^i z_i^i dm_i & \int y_i^i z_i^i dm_i & \int z_i^{i2} dm_i & \int z_i^i dm_i \\ \int x_i^i dm_i & \int y_i^i dm_i & \int z_i^i dm_i & \int dm_i \end{bmatrix} \\
&= \begin{bmatrix} \frac{-S_{ixx}^2+S_{iyy}^2+S_{izz}^2}{2} & S_{ixy}^2 & S_{ixz}^2 & \bar{x}_i^i \\ S_{ixy}^2 & \frac{S_{ixx}^2-S_{iyy}^2+S_{izz}^2}{2} & S_{iyz}^2 & \bar{y}_i^i \\ S_{ixz}^2 & S_{iyz}^2 & \frac{S_{ixx}^2+S_{iyy}^2-S_{izz}^2}{2} & \bar{z}_i^i \\ \bar{x}_i^i & \bar{y}_i^i & \bar{z}_i^i & 1 \end{bmatrix} m_i \\
&= \begin{bmatrix} \frac{-I_{ixx}+I_{iyy}+I_{izz}}{2} & I_{ixy} & I_{ixz} & m_i\bar{x}_i^i \\ I_{ixy} & \frac{I_{ixx}-I_{iyy}+I_{izz}}{2} & I_{iyz} & m_i\bar{y}_i^i \\ I_{ixz} & I_{iyz} & \frac{I_{ixx}+I_{iyy}-I_{izz}}{2} & m_i\bar{z}_i^i \\ m_i\bar{x}_i^i & m_i\bar{y}_i^i & m_i\bar{z}_i^i & m_i \end{bmatrix} \qquad (2.24)
\end{aligned}
$$

where $\bar{\mathbf{r}}_i^i = [\bar{x}_i^i \quad \bar{y}_i^i \quad \bar{z}_i^i \quad 1]'$ is the center of mass vector of link i from the ith link coordinate frame and S_{ixy} is called the *radius of gyration* of link i about $X_i - Y_i$ axes - *Inertia tensor* I_i and *first moments of body* are defined as:

- **Inertia Tensor I_i**

  1. Moments of Inertia

$$I_{ixx} = \int \left(y_i^{i2} + z_i^{i2} \right) dm_i$$

$$I_{iyy} = \int \left(x_i^{i2} + z_i^{i2} \right) dm_i$$

$$I_{izz} = \int \left(x_i^{i2} + y_i^{i2} \right) dm_i$$

  2. Cross-Products of Inertia (Symmetry : $I_{ixy} = I_{iyx}$)

$$I_{ixy} = \int x_i^i y_i^i dm_i$$

$$I_{ixz} = \int x_i^i z_i^i dm_i$$

$$I_{iyz} = \int y_i^i z_i^i dm_i$$

- **First Moments of Body**

$$m\bar{x}_i^i = \int x_i^i dm_i$$

$$m\bar{y}_i^i = \int y_i^i dm_i$$

$$m\bar{z}_i^i = \int z_i^i dm_i$$

Hence, the total kinematic energy of the manipulator from equation 2.23 is:

$$\begin{aligned} K &= \sum_{i=1}^{n} K_0^i \\ &= \sum_{i=1}^{n} \left\{ \frac{1}{2} Tr \left[\sum_{j=1}^{i} \sum_{k=1}^{i} \frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^i}{\partial q_k}' \dot{q}_j \dot{q}_k \right] \right\} \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{i} Tr \left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^i}{\partial q_k}' \right] \dot{q}_j \dot{q}_k \end{aligned} \tag{2.25}$$

The total kinetic energy K is *scalar*, and the off-diagonal elements of J_i are zero because the classical cross-products of inertia are assumed to be zero [13].

2.6.3 Potential Energy of a Robot Manipulator

Let the total potential energy of a robot manipulator be P and each of ith link's potential energy be P_i:

$$\begin{aligned} P_i &= -m_i \mathbf{g} (\bar{\mathbf{r}}_0^i) \\ &= -m_i \mathbf{g} (A_0^i \bar{\mathbf{r}}_i^i) \qquad i = 1, 2, \ldots, n \end{aligned} \tag{2.26}$$

where $\bar{\mathbf{r}}_0^i = [\bar{x}_0^i \ \ \bar{y}_0^i \ \ \bar{z}_0^i]'$ and $\bar{\mathbf{r}}_i^i = [\bar{x}_i^i \ \ \bar{y}_i^i \ \ \bar{z}_i^i]'$.

The total potential energy of the robot manipulator can be obtained by summing all the potential energies in each link:

$$\begin{aligned} P &= \sum_{i=1}^{n} P_i \\ &= \sum_{i=1}^{n} -m_i \mathbf{g}(A_0^i \bar{\mathbf{r}}_i^i) \end{aligned} \tag{2.27}$$

where $\mathbf{g} = [g_x,\ g_y,\ g_z,\ 0]$ is a gravity row vector expressed in the base coordinate system. For a level system, $\mathbf{g} = [0,\ 0,\ -g,\ 0]$ and g is the gravitational constant ($g = 9.8062\ m/s^2$).

2.6.4 L-E Dynamics Equations of a Robot Manipulator

The Lagrangian function $\mathcal{L} = K - P$ can now be formed from the kinetic energy and the potential energy, i.e.,

$$\begin{aligned} \mathcal{L} &= \{K\} - \{P\} \\ &= \left\{ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{i} Tr \left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^{i\,T}}{\partial q_k} \right] \dot{q}_j \dot{q}_k \right\} - \left\{ \sum_{i=1}^{n} -m_i \mathbf{g}(A_0^i \bar{\mathbf{r}}_i^i) \right\} \end{aligned} \tag{2.28}$$

We now obtain the dynamics equations of a robot manipulator from below the Lagrange-Euler equation.

$$\tau_i = \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} \qquad i = 1, \ldots, n \tag{2.29}$$

Evaluation of $\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_p}\right)$

Let's perform first differentiation of $\frac{\partial \mathcal{L}}{\partial \dot{q}_p}$:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \dot{q}_p} &= \frac{\partial}{\partial \dot{q}_p}\left\{\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right]\dot{q}_j\dot{q}_k + \sum_{i=1}^{n} m_i \mathbf{g}(A_0^i \bar{\mathbf{r}}_i^i)\right\} \\
&= \frac{1}{2}\sum_{i=1}^{n}\frac{\partial}{\partial \dot{q}_p}\left\{\sum_{j=1}^{i}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right]\dot{q}_j\dot{q}_k\right\} \\
&\quad + \frac{\partial}{\partial \dot{q}_p}\left\{m_1\mathbf{g}A_0^1\bar{\mathbf{r}}_1^1 + m_2\mathbf{g}A_0^2\bar{\mathbf{r}}_2^2 + \cdots + m_n\mathbf{g}A_0^n\bar{\mathbf{r}}_n^n\right\} \\
&= \frac{1}{2}\sum_{i=1}^{n}\left\{\sum_{j=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^{i\prime}}{\partial q_p}\right]\dot{q}_j + \frac{1}{2}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_p} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right]\dot{q}_k\right\} \\
&\quad + 0 \qquad \left(\text{since } \frac{\partial A_0^i}{\partial \dot{q}_p} = 0\right) \\
&= \frac{1}{2}\sum_{i=1}^{n}\underbrace{\sum_{j=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^{i\prime}}{\partial q_p}\right]\dot{q}_j}_{(a)} + \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_p} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right]\dot{q}_k
\end{aligned}$$

By changing the dummy index j to k at (a),

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^{i\prime}}{\partial q_p}\right]\dot{q}_k + \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{i}\underbrace{Tr\left[\frac{\partial A_0^i}{\partial q_p} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right]}_{(b)}\dot{q}_k$$

Since $Tr\left[\frac{\partial A_0^i}{\partial q_p} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right] = Tr\left[\frac{\partial A_0^i}{\partial q_p} J_i \frac{\partial A_0^{i\prime}}{\partial q_k}\right]' = Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^{i\prime}}{\partial q_p}\right]$ at (b),

$$= \begin{cases} \displaystyle\sum_{i=1}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^{i\prime}}{\partial q_p}\right]\dot{q}_k & \text{for } p \le i \\ 0 & \text{for } p > i \text{ since } \frac{\partial A_0^i}{\partial p} = 0 \end{cases} \tag{2.30}$$

We obtain:

$$\frac{\partial \mathcal{L}}{\partial \dot{q}_p} = \sum_{i=p}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^{i\prime}}{\partial q_p}\right]\dot{q}_k \tag{2.31}$$

Hence,

$$\begin{aligned}
\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_p}\right) &= \frac{d}{dt}\left(\sum_{i=p}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^i}{\partial q_p}^{T}\right]\dot{q}_k\right) \\
&= \sum_{i=p}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^i}{\partial q_p}^{T}\right]\ddot{q}_k \\
&\quad + \sum_{i=p}^{n}\sum_{k=1}^{i}\sum_{m=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_k \partial q_m} J_i \frac{\partial A_0^i}{\partial q_p}^{T}\right]\dot{q}_k\dot{q}_m \\
&\quad + \sum_{i=p}^{n}\sum_{k=1}^{i}\sum_{m=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_p \partial q_m} J_i \frac{\partial A_0^i}{\partial q_k}^{T}\right]\dot{q}_k\dot{q}_m \qquad (2.32)
\end{aligned}$$

Evaluation of $\frac{\partial \mathcal{L}}{\partial q_p}$

The last term of Lagrange-Euler equation is evaluated, that is :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial q_p} &= \frac{\partial}{\partial q_p}\left(\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_j} J_i \frac{\partial A_0^i}{\partial q_k}^{T}\right]\dot{q}_j\dot{q}_k + \sum_{i=1}^{n} m_i \mathbf{g} A_0^i \bar{\mathbf{r}}_i^i\right) \\
&= \frac{1}{2}\sum_{i=p}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_j \partial q_p} J_i \frac{\partial A_0^i}{\partial q_k}^{T}\right]\dot{q}_j\dot{q}_k \\
&\quad + \sum_{i=p}^{n}\sum_{j=1}^{i}\underbrace{\sum_{k=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_k \partial q_p} J_i \frac{\partial A_0^i}{\partial q_j}^{T}\right]}_{(a)}\dot{q}_j\dot{q}_k \\
&\quad + \sum_{i=p}^{n} m_i \mathbf{g} \frac{\partial A_0^i}{\partial q_p}\bar{\mathbf{r}}_i^i
\end{aligned}$$

By interchanging the dummy indices of j and k at (a),

$$= \sum_{i=p}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_p \partial q_j} J_i \frac{\partial A_0^i}{\partial q_k}^{T}\right]\dot{q}_j\dot{q}_k + \sum_{i=p}^{n} m_i \mathbf{g} \frac{\partial A_0^i}{\partial q_p}\bar{\mathbf{r}}_i^i$$

By changing j to m and then swapping $\sum_{m=1}^{i}$ and $\sum_{k=1}^{i}$,

$$= \sum_{i=p}^{n}\sum_{k=1}^{i}\sum_{m=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_p \partial q_m} J_i \frac{\partial A_0^i}{\partial q_k}^{T}\right]\dot{q}_k\dot{q}_m + \sum_{i=p}^{n} m_i \mathbf{g} \frac{\partial A_0^i}{\partial q_p}\bar{\mathbf{r}}_i^i \qquad (2.33)$$

Evaluation of $\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_p}\right) - \frac{\partial \mathcal{L}}{\partial q_p}$

From equations 2.32 and 2.33, we obtain: the third term of $\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_p}\right)$ in equation 2.32 cancels the first term of $\frac{\partial \mathcal{L}}{\partial q_p}$ in equation 2.33.

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_p}\right) - \frac{\partial \mathcal{L}}{\partial q_p} &= \sum_{i=p}^{n}\sum_{k=1}^{i} Tr\left[\frac{\partial A_0^i}{\partial q_k} J_i \frac{\partial A_0^i}{\partial q_p}'\right]\ddot{q}_k \\ &+ \sum_{i=p}^{n}\sum_{k=1}^{i}\sum_{m=1}^{i} Tr\left[\frac{\partial^2 A_0^i}{\partial q_k \partial q_m} J_i \frac{\partial A_0^i}{\partial q_p}'\right]\dot{q}_k\dot{q}_m \\ &- \sum_{i=1}^{n} m_i \mathbf{g} \frac{\partial A_0^i}{\partial q_p}\bar{\mathbf{r}}_i^i \end{aligned} \tag{2.34}$$

Finally, we obtain the *dynamics equation* of a robot manipulator by changing dummy summation indices p to i and i to j:

$$\begin{aligned} \tau_i &= \sum_{j=i}^{n}\sum_{k=1}^{j} Tr\left[\frac{\partial A_0^j}{\partial q_k} J_j \frac{\partial A_0^j}{\partial q_i}'\right]\ddot{q}_k \\ &+ \sum_{j=i}^{n}\sum_{k=1}^{j}\sum_{m=1}^{j} Tr\left[\frac{\partial^2 A_0^j}{\partial q_k \partial q_m} J_j \frac{\partial A_0^j}{\partial q_i}'\right]\dot{q}_k\dot{q}_m \\ &- \sum_{j=i}^{n} m_j \mathbf{g} \frac{\partial A_0^j}{\partial q_i}\bar{\mathbf{r}}_j^j \end{aligned} \tag{2.35}$$

2.6.5 Computational Simplification of Dynamics Equation using $Q - matrix$

The computation of the *matrix partial derivatives* in equation 2.35 is very time consuming. The calculations can be made faster by first noticing that transformation matrix A_{i-1}^i is a function of the generalized coordinate q_i only. So, the computation of partial derivative $\partial A_{i-1}^i/\partial q_i$ for serial link manipulators can be converted to a multiplication of matrices [1].

If we define $Q_i - matrix$ for a revolute joint i as below:

$$Q_i \equiv \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.36}$$

Then,

$$\frac{\partial A_{i-1}^{i}}{\partial q_i} = Q_i A_{i-1}^{i} \tag{2.37}$$

For example:

$$\begin{aligned} \frac{\partial A_{i-1}^{i}}{\partial \theta_i} &= \begin{bmatrix} -\sin\theta_i & -\cos\alpha_i\cos\theta_i & \sin\alpha_i\cos\theta_i & -a_i\sin\theta_i \\ \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\cos\theta_i & a_i\cos\theta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= Q_i A_{i-1}^{i} \end{aligned}$$

Hence, for $i = 1, 2, \ldots, n$:

$$\begin{aligned} \frac{\partial A_0^i}{\partial q_j} &= \frac{\partial}{\partial q_j}\left(\underbrace{A_0^1 A_1^2 \cdots A_{j-2}^{j-1}}\ \underbrace{A_{j-1}^{j}}\ \underbrace{A_j^{j+1} \cdots A_{i-1}^{i}}\right) \\ &= \overbrace{A_0^1 A_1^2 \ldots A_{j-2}^{j-1}}\ \overbrace{Q_j A_{j-1}^{j}}\ \overbrace{A_j^{j+1} \cdots A_{i-1}^{i}} \\ &= \begin{cases} A_0^{j-1} Q_j A_{j-1}^{i} & \text{for } 1 \le j \le i \\ 0 & \text{for } j > i \end{cases} \end{aligned} \tag{2.38}$$

In order to simplify partial derivative notations of dynamics equation 2.35, we define the $U-matrix$ as follows:

$$U_{ij} \equiv \frac{\partial A_0^i}{\partial q_j} = \begin{cases} A_0^{j-1} Q_j A_{j-1}^i & \text{for } j \le i \\ 0 & \text{for } j > i \end{cases} \tag{2.39}$$

$$U_{ijk} \equiv \frac{\partial U_{ij}}{\partial q_k}$$

$$= \frac{\partial^2 A_0^i}{\partial q_k \partial q_j} = \begin{cases} A_0^{j-1} Q_j A_{j-1}^{k-1} Q_k A_{k-1}^i & \text{for } i \ge k \ge j \\ A_0^{k-1} Q_k A_{k-1}^{j-1} Q_j A_{j-1}^i & \text{for } i \ge j \ge k \\ 0 & \text{for } i < j \text{ or } i < k \end{cases} \tag{2.40}$$

For example, for a manipulator with all revolute joints $i = j = 1$ and $q_1 = \theta_1$,

$$U_{111} = \frac{\partial U_{11}}{\partial \theta_1} = \frac{\partial}{\partial \theta_1}\left(Q_1 A_0^1\right) = Q_1 Q_1 A_0^1$$

Equation 2.40 can be interpreted as the interaction effects of the motion of joint j and joint k on all the points in link i. Hence, dynamics equation 2.35 is rewritten without partial derivative operations by using the $U-$*matrix*:

$$\tau_i = \sum_{j=i}^{n} \sum_{k=1}^{j} Tr[U_{jk} J_j U'_{ji}] \ddot{q}_k + \sum_{j=i}^{n} \sum_{k=1}^{j} \sum_{m=1}^{j} Tr[U_{jkm} J_j U'_{ji}] \dot{q}_k \dot{q}_m - \sum_{j=i}^{n} m_j \mathbf{g} U_{ji} \bar{\mathbf{r}}_j^j \qquad \text{for } i = 1, \ldots, n \tag{2.41}$$

The above equation can be expressed in a much simpler form as:

$$\tau_i = \sum_{k=1}^{n} D_{ik} \ddot{q}_k + \sum_{k=1}^{n} \sum_{m=1}^{n} c_{ikm} \dot{q}_k \dot{q}_m + G_i \qquad i = 1, \ldots, n \tag{2.42}$$

Or in a matrix form:

$$\boldsymbol{\tau}(t) = \mathbf{D}(\mathbf{q}(t)) \ddot{\mathbf{q}}(t) + \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t)) + \mathbf{G}(\mathbf{q}(t)) \tag{2.43}$$

where

$\boldsymbol{\tau}(t) = n \times 1$ generalized torque vector applied at joint $i = 1, \ldots, n$, and can be expressed as:

$$\boldsymbol{\tau}(t) = [\tau_1(t),\ \tau_2(t), \ldots, \tau_n(t)]'$$

$\mathbf{q}(t) = n \times 1$ vector of the joint variables of the robot manipulator and can be expressed as:

$$\mathbf{q}(t) = [q_1(t),\ q_2(t), \ldots, q_n(t)]'$$

$\dot{\mathbf{q}}(t) = n \times 1$ vector of the joint velocity of the robot manipulator and can be expressed as:

$$\dot{\mathbf{q}} = [\dot{q}_1(t),\ \dot{q}_2(t), \ldots, \dot{q}_n(t)]'$$

$\ddot{\mathbf{q}}(t) = n \times 1$ vector of the acceleration of the joint variable $\mathbf{q}(t)$ and can be expressed as:

$$\ddot{\mathbf{q}} = [\ddot{q}_1(t),\ \ddot{q}_2(t), \ldots, \ddot{q}_n(t)]'$$

$\mathbf{D}(\mathbf{q}) = n \times n$ inertial acceleration-related symmetric matrix whose elements are:

$$D_{ik} = \sum_{j=\max(i,k)}^{n} Tr[U_{jk} J_j U'_{ji}] \qquad i, k = 1, \ldots, n$$

$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = n \times 1$ nonlinear Coliolis and centrifugal force vector whose elements are:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = [C_1, C_2, \ldots, C_n]'$$

where

$$\begin{aligned} C_i &= \sum_{k=1}^{n} \sum_{m=1}^{n} c_{ikm} \dot{q}_k \dot{q}_m \qquad i = 1, \ldots, n \\ c_{ikm} &= \sum_{j=\max(i,k,m)}^{n} Tr[U_{jkm} J_j U'_{ji}] \quad i, k, m = 1, \ldots, n \end{aligned}$$

$\mathbf{G}(\mathbf{q}) = n \times 1$ gravity loading force vector whose elements are:

$$\mathbf{G}(\mathbf{q}) = [G_1, G2, \ldots, G_n]'$$

where

$$G_i = \sum_{j=i}^{n} (-m_j \mathbf{g} U_{ji} \bar{\mathbf{r}}_j^j) \qquad i = 1, \ldots, n$$

2.7 Customized Closed-form of Dynamics Equation for Six d-o-f PUMA Robot

For controlling the robot manipulator or simulating its behavior on a computer, the dynamics coefficients of in equation 2.43, that is, $\mathbf{D}(\mathbf{q}(t)), \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ and $\mathbf{G}(\mathbf{q}(t))$, need to be computed. However, the algebraic manipulations leading to the complete dynamic model for a robot manipulator become tedious and time-consuming as the number n of d-o-f increases, although the formulation of equation 2.43 is inherently straightforward [10, 2, 18]. So, complete dynamics equations are seldom presented for robot manipulators with more than three d-o-f although formulations for generating complete dynamic robot models have been shown in the literature [10, 3]. The following section is to show how to build the customized closed-form dynamics equation based on Lagrange-Euler dynamics model of equation 2.43.

2.7.1 Customizing Dynamics Equation

In contrast to generalized-purpose algorithms, the practical problem with customized algorithms is that a different algorithm is required for a specified robot manipulator to reduce the computational requirements of manipulator dynamics for real-time control.

Computational savings of customized algorithms stem from the kinematic and dynamic structure of the manipulator and systematic organization of the symbolic model when the symbolically processing computer program is used [9, 11].

For example, Algebraic Robot Modeler(ARM) [11] resolves this problem as follows: The kinematic and dynamic structure of the manipulator is exploited during the *off-line* symbolic modeling as additions of zero and multiplications by ±1/zero are canceled algebraically. Upon completing the modeling stage, ARM applies a few elementary grouping and factoring rules to the algebraic expressions in closed-form dynamic robot models to remove repetitive calculations within and across equations.

On the other hand, the dynamic coefficients D_{ik} and c_{ikm} in equation 2.42 exhibit the symmetric reflective coupling properties [8, 16].

- Symmetry: $D_{ik} = D_{ki}$ and $c_{ikm} = c_{imk}$.
- Reflective coupling: $c_{ikm} = -c_{mki}$ for $k \leq i$ and m.

The dynamics coefficients D_{ik}, c_{ikm}, and G_i depend on the constant geometrical, inertial, and gravitational parameters of the robot manipulator. The symbolically processing software, which produces the closed-form of the dynamics robot equations, accepts these constant parameters as input, either symbolically or numerically at the user's option.

The following section describes the parameters shown in the [13] with which ARM generates the complete customized closed-form of the dynamics equation of the PUMA 600 robot manipulator. This dynamics equation is useful for simulating the motion of the physical robot and thus, is used in simulating the plant of the PUMA robot manipulator on a computer in this thesis.

2.7.2 Parameters of PUMA 600

The six d-o-f revolute PUMA robot has parallel/perpendicular joint axes, a diagonal link inertia tensor, a sparse center-of-mass vector, and the six link coordinate frames assigned in 2.1. Physical parameter values for the PUMA 600 are shown as follows:

Kinematic Link Parameters

The transformation matrix A_{i-1}^{i} for the dynamics equation is evaluated with Table 2.2.

$$A_{i-1}^{i} = \left[\begin{array}{ccc|c} \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

joint $i =$	α_i°	θ_i°	d_i	a_i
1	−90	θ_1	0	0
2	0	θ_2	0	a_2=43.2cm
3	90	θ_3	d_3=12.5cm	a_3=1.9cm
4	−90	θ_4	d_4=43.2cm	0
5	90	θ_5	0	0
6	0	θ_6	0	0

Table 2.2: Kinematic Link Parameters for the PUMA 600

Center-of-Mass and Relative Link Mass

We assume that all masses are nonzero and focus on coordinate $\mathbf{r}_i^i$ in each link i. Thus, the center-of-mass of link i in the ith coordinate frame is denoted by $\bar{\mathbf{r}}_i^i = [\bar{x}_i^i \;\; \bar{y}_i^i \;\; \bar{z}_i^i]'$ and values of center-of-mass and relative link mass are shown in Table 2.3.

Link	$\bar{x}(cm)$	$\bar{y}(cm)$	$\bar{z}(cm)$	Relative Mass(m_i/m_6)
1	0	0	8	33.5
2	−21.6	0	21.75	77.3
3	0	0	21.6	36.3
4	0	2	0	8.95
5	0	0	2	2.39
6	0	0	1	1

Table 2.3: Center-Of-Mass and Relative Link Mass

The total weight of the PUMA 500 series manipulator is 120 pounds [19]. After subtracting the weight of base, the remaining mass of six links is assumed to be about 60 pounds. By equating the relative link mass shown in the [13], each link mass is shown in the Appendix A.

Gravitational Acceleration Vector

The gravitational acceleration vector **g** points in the negative Z_0−axis direction and it is denoted as below:

$$\mathbf{g} = \begin{bmatrix} 0 \\ 0 \\ -g \\ 0 \end{bmatrix}$$

where $g = 9.8062\ m/s^2$.

Pseudo-Inertia Matrix

Pseudo-inertia matrix J_i is assumed to be diagonal in [12, 13]. So off-diagonal pseudo-inertias are zero, and the main diagonal is composed of radii-of-gyration.

$$J_i = \begin{bmatrix} J_{ixx} & 0 & 0 & 0 \\ 0 & J_{iyy} & 0 & 0 \\ 0 & 0 & J_{izz} & 0 \\ 0 & 0 & 0 & m_i \end{bmatrix}$$

where

$$\begin{aligned} J_{ixx} &= \frac{1}{2}m_i(-S^2_{ixx} + S^2_{iyy} + S^2_{izz}) \\ J_{iyy} &= \frac{1}{2}m_i(S^2_{ixx} - S^2_{iyy} + S^2_{izz}) \\ J_{izz} &= \frac{1}{2}m_i(S^2_{ixx} + S^2_{iyy} - S^2_{izz}). \end{aligned}$$

The experimental values [13] of radii-of-gyration for the PUMA 600 are shown in Table 2.4.

Link	$S^2_{xx}(cm^2)$	$S^2_{yy}(cm^2)$	$S^2_{zz}(cm^2)$
1	451	451	57.9
2	565.7	1847	1408
3	672.8	679.1	36
4	31.6	21.1	31.6
5	6.9	11.2	6.9
6	33.8	33.8	0.911

Table 2.4: Radii-of-Gyration

2.8 Digital Simulation of PUMA 600 Motion

Simulation of physical manipulator motion implies the forward dynamics problem; that is, given certain torques/forces, the joint acceleration is solved. This requires solving the inverse dynamics equation 2.43 as below:

$$\ddot{\Theta}(k) = \mathbf{D}^{-1}(\Theta(k))\left[\boldsymbol{\tau}(k) - \mathbf{C}(\Theta(k), \dot{\Theta}(k)) - \mathbf{G}(\Theta(k))\right] \tag{2.44}$$

For the recursive calculation of the equation 2.44 at each time k, the initial values of $\boldsymbol{\tau}(0)$, $\Theta(0)$ and $\dot{\Theta}(0)$ should be given by controller.

The kinematic structural parameters to compute $\mathbf{D}(\cdot)$, $\mathbf{C}(\cdot)$ and $\mathbf{G}(\cdot)$ in equation 2.44 are given in Table 2.1. The inversion of a inertia matrix, $\mathbf{D}(\cdot)$, is done by using the well known lower/upper triangular(LU) decomposition and backsubstitution. For some applications on singular matrices, the LU decomposition method substitutes tiny value(1.0e-20) for a zero pivot element [14].

2.8.1 Measured Angular Velocity and Position with Measurement Error

Velocity and position are computed from the acceleration equation 2.44 by the Euler's method.

$$\dot{\Theta}(k+1) = \dot{\Theta}(k) + \ddot{\Theta}(k)T_s \tag{2.45}$$

$$\Theta(k+1) = \Theta(k) + \dot{\Theta}(k)T_s + \frac{1}{2}\ddot{\Theta}(k)T_s^2 \tag{2.46}$$

The sampling period $T_s = 0.01$ second is used in the sense that breaking continuous times into increments would be a reasonable approximation. The measurements of joint velocity $\dot{\Theta}^m(k)$ is generated by superimposing sample values of white Gaussian noise process with zero-mean and variance σ^2, which accounts for measurement error.

$$\dot{\Theta}^m(k) = \dot{\Theta}(k) + \mathcal{N}(k) \tag{2.47}$$

where $\dot{\Theta}^m(k)$ is a (6×1) measured velocity vector, $\dot{\Theta}(k)$ is a (6×1) pure velocity vector from the inverse dynamics equations and $\mathcal{N}(k)$ is a (6×1) measurement error vector whose elements have independent samples of white Gaussian noise process with $\sigma^2 = 0.01$. Here, only measurements of joint velocity are assumed to be available because the measurement of joint position can be computed by integrating the joint velocity measurements of equation 2.47 in the controller unit, provided that the initial position $\Theta^m(0)$ is known:

$$\begin{aligned}
\Theta^m(k) &= \Theta^m(k-1) + \dot{\Theta}^m(k-1)T_s + \frac{1}{2}\ddot{\Theta}^m(k-1)T_s^2 \\
&= \Theta^m(k-1) + \dot{\Theta}^m(k-1)T_s + \frac{1}{2}\left[\frac{\dot{\Theta}^m(k) - \dot{\Theta}^m(k-1)}{T_s}\right]T_s^2 \\
&= \Theta^m(k-1) + \frac{1}{2}\dot{\Theta}^m(k-1)T_s + \frac{1}{2}\dot{\Theta}^m(k)T_s \qquad (2.48)
\end{aligned}$$

where $\Theta^m(k-1)$, $\dot{\Theta}^m(k-1)$ and $\dot{\Theta}^m(k)$ are all available at instance k.

When a self-tuning controller is designed using a model with position output, difficulties in tracking may be experienced when the model for the positional output is of low order, that is, when the model for the position is not sufficiently rich in the frequency content. These difficulties may often be circumvented by constructing a time-series model for the joint velocities [6].

2.8.2 About Generation of Gaussian Noise

The white Gaussian zero-mean process with variance σ^2 is generated with random numbers from a random number generating function in the program. However, the random number generating function requires *different* **seed** as an argument for generating different pattern of numbers in each execution of program. Generally this value of the seed is fed into the program manually. In this thesis, different seeds are automatically generated into the program by reading the present time from computer clock and then multiplied hours, minutes and seconds by different weights as follows:

$$\text{seed} = (hour \times 10000) + (minute \times 100) + second \qquad (2.49)$$

where hours, minutes and seconds are present time read by the time-reading function in program. With these different seeds, each execution of the main program uses different measurement error patterns of the Gaussian noise process.

2.9 Remarks

The customized closed-form of the dynamics equation for six d-o-f PUMA robot can be realized on commercially available processors, and is applicable to real-time control algorithms which require the on-line evaluation of manipulator dynamics. MATLAB programs in the Appendix A shows the customized closed-form of dynamics equations based on the L-E dynamics.

Bibliography

[1] A. K. Bejczy. Robot arm dynamics and control. Technical Report Technical Memo 33-669, Jet Propulsion Laboratory, Pasadena, Calif., 1974.

[2] A. K. Bejczy. Dynamic analysis for robot arm control. *Proc. 1983 American Control Conf.*, pages 503–504, 1983. San Francisco, CA.

[3] M. Brady. *Robot Motion: Planning and Control.* MIT Press, Cambridge, MA, 1982.

[4] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanics based on matrices. *J. of Applied Mechanics*, pages 215–221, June 1955.

[5] R. L. Huston and F. A. Kelly. The development of equations of equations of motion of single-arm robots. *IEEE Trans., Systems, Man, and Cybernetics*, SMC-12(no. 3):pp.259–266, 1982.

[6] A.J. Koivo. *Fundamentals for Control of Robotic Manipulators.* John Wiley & Sons Inc., 1989.

[7] C. S. Lee. Robotic arm kinematics dynamics and control. *Computer, IEEE*, pages 62–80, December 1982.

[8] R. A. Lewis. Autonomous manipulation on a robot: Summary of manipulator software functions. Technical Report Tech. Memo. 33-69, Jet Propulsion Lab., Pasadena, CA, March 1974.

[9] J. J. Murray and C. P. Neuman. Arm: An algebraic robot dynamic modeling program. *Proc. of 1984 IEEE International Conference on Robotics*, pages 103–114, 1984. Atlanta.

[10] C. P. Neuman and J. J. Murray. Computational robot dynamics: Foundations and applications. *J. Robotic Syst.*, vol.2:pp.425–452, Winter 1985.

[11] C. P. Neuman and J. J. Murray. The complete dynamic model and customized algorithms of the puma robot. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-17(no. 4):pp.635–644, July/August 1987.

[12] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, MA, 1981.

[13] R. P. Paul, M. Rong, and H. Zhang. The dynamics of the puma manipulator. *Proc. 1983 American Control Conf.*, pages 491–496, June 1983. San Francisco, CA.

[14] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[15] Symbolic Computation Group, The University of Waterloo, Ont., Canada. *Maple V.*

[16] T. D. Tourassis and C. P. Neuman. The inertial characteristics of dynamic robot models. In *Mechanism and Machine Theory*, volume vol.20, pages pp.41–52, 1985.

[17] J. J. Uicker. *On the Dynamic Analysis of Spatial Linkages Using 4x4 Matrices.* PhD thesis, Northwestern University, 1965.

[18] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Trans., ASME, J. Systems, Measurement and Control*, vol.104:pp.205–211, 1982.

[19] Westinghouse Company. *500 Series Equipment Manual for VAL* II *and VAL PLUS Operating Systems*, 1985.

Chapter 3

Artificial Neural Networks

3.1 Introduction

This chapter describes the general theory about Artificial Neural Networks (ANNs). ANNs try to mimic the nerve system in a mammalian brain, which is a highly complex, nonlinear and parallel information processing system, into a mathematical model. The application of ANN promises a high computation rate by the massive parallelism, great degree of robustness (or fault-tolerance) due to the distributed representation, and can result in the ability of adaptation, learning, diagnosis and generalization for the system being investigated [34],[14].

Parallel Processing

The brain and ANNs process information in a similar manner. The topology of neural networks enables them to process large dimensional information simultaneously by massive calculation with a specific parallel computing system. This is hundreds of times faster than serial architecture computers. (In conventional single-processor Von Neumann computers, the speed of computation is limited by the propagation delay of the transistors.) For example, consider the amount of computation needed to process a single visual image. If one restricts the image resolution to 1000 × 1000 receptors, several million computations are performed in order that objects in the image are identified. Even at the nanosecond speeds of modern computers, this task can require several seconds in a conventional computer. And even then the recognition may be marginal. In contrast, biological visual systems compute such tasks in milliseconds. The reason for this disparity in performance is, of course, due to the massively parallel computations being performed in the brain. At any one time, tens of thousands of computations are being performed simultaneously within biological neural network. However, ANNs do lack some characteristics, including the ability to forget[61] although they share many distinct features with biological networks.

Mapping

Neural networks can perform a mapping between an input and output space, and synthesize an associative memory that retrieves the appropriate output when presented with the input and generalizes when presented with new inputs[45]. The above ability to learn arbitrary functions from a set of training examples is one of the most important characteristics of some ANNs. This capability covers a diverse range of applications. In fact, one could argue that most ANN applications fall under the general heading of functional mappings, where an ANN learns to transform an n-dimensional input vector to an m-dimensional output vector according to some criteria. For example, a neural network can be regarded as a black-box that transforms input vector $\mathbf{x}$ from an n-dimensional space to an output vector $\mathbf{y}$ in m-dimensional space $\mathbf{G} : \mathbf{x} \rightarrow \mathbf{y}$. The types of mappings $\mathbf{G}$ which a network can approximate depends on the particular ANN architecture. In general, the mapping $\mathbf{G}$ will be either autoassociative (mapping to an original pattern from a noisy or partially given input pattern) or heteroassociative (mapping from an input pattern to a different output pattern). Meanwhile, certain multilayer feedforward networks can approximate almost any reasonably well behaved functions $\mathbf{G}$ to any desired degree of accuracy. In order to implement such a network, however, it may be necessary to use an unwieldy number of neurons if very high accuracy is desired.

Generalization By Learning

Generalization can be regarded as the process of describing the whole from some of the parts, reasoning from the specific to the general case, or defining a class of objects from a knowledge of one or more instances. Generalization is an essential part of learning as it permits us to remember facts that apply only to individual members of the class. It serves as an efficient mode of memorization and storage. ANNs generalize when they compute or recall full patterns from partial or noisy input patterns, when they recognize or classify objects not previously trained on, or when they predict new outcomes from past behaviors. The ability to classify objects not previously trained on is a form of interpolation between trained patterns. The ability to predict from past behaviors is a form of extrapolation. Both of these types of mappings are a form of generalization.

Robust Performance

If a certain system continues to perform well when part of it is disabled or when presented with noisy data, that can be typically robust as a computing system. ANNs establish such features that if one or a few neurons fail during operation, the distributed storage of information over a set of network neurons ensures a graceful degradation in the networks' performance. None of the information is completely lost and the neural networks are still capable of fault-tolerant operation. This is possible because the *knowledge* stored in an ANN is distributed over many neurons and interconnections, not just a single or a few units. Consequently, concepts or mappings stored in an ANN have some degree of redundancy built in through this distribution of knowledge. This aspect of ANNs is sometimes called *robust performance* by fault-tolerance. This is in sharp contrast to conventional computers, where the loss of a single transistor or other component in a serial (Von Neumann) computer can result in complete system failure. Such systems are most intolerant of faults. One would expect the human brain to exhibit similar characteristics to ANNs. A portion of the brain can be damaged or removed without seriously affecting the performance of an individual.

3.1.1 Neural Networks Application

ANNs can perform functional approximation and signal filtering operations that are beyond optimal linear techniques because of their nonlinear nature. This interest stems from a recognition of the fact that the dynamics of many phenomena cannot be accurately described using linear models. Important ANN architectures have nonlinear dynamics. Their behavior cannot be fully appreciated without an understanding of nonlinear dynamical systems theory. Many ANN applications are related to the prediction of nonlinear systems behavior. For example, ANNs have been shown to be effective as tools in forecasting the future movement of a time series which is known to be driven by nonlinear dynamics. ANNs themselves offer a promising approach to better understand, classify and model nonlinear systems. They can be used effectively to estimate system parameters and thereby help to characterize these systems. Typical applications are given bellow.

Control

ANNs have been used effectively in learning to control robotic manipulators, outdoor mobile robots including driverless driving tasks (autonomous land vehicles). They have also been used to efficiently control the positioning of huge electrodes in electric arc furnaces used by steel-making companies, saving the companies millions of dollars through reduced electricity consumption and extended life of costly equipment. They have been used to control and optimize chemical plant processes saving companies huge sums of money through better process control and material usage. Dozens of consumer products, especially those manufactured by Japanese companies, incorporate neural network controllers including induction cookers, microwave ovens, air conditioners, clothes dryers, photocopy machines, word processors as well as others. ANNs have been used both independently and in conjunction with fuzzy logic controllers to improve the performance of a product. The range of applications in the area of control seems unlimited.

Multi Sensor Data Fusion

Sensor data fusion is the process of combining data from multiple sensors in order to derive more information through individual sources. The fusion process includes detection, association, correlation, estimation, and combination of data to achieve identity estimation and timely assessment of situations. The sensor technology in 1980s has led to rapid expansion of multisensor fusion applications, including military, process control, monitoring, robotics, diagnostics and others.

The most powerful example of large scale multisensor fusion is found in human and other biological systems. Humans apply fusion of body's sensory data of touch, sight, sound and scent to gain meaningful perception of the environment. Hundreds of thousands of sensors are collecting data in real-time for fusion and processing through successively higher levels of abstraction. The nervous system performs the fusion of sensory data through its massively interconnected network of neurons.

Artificial neural networks offer great promise in multisensor data fusion applications for the same reasons that biological systems are so successful at these tasks. ANN architectures and processing capabilities show a kind of data fusion to achieve desired mapping of input to output signals.

Optimization

ANNs have been used for a number of problems that require finding an optimal or near optimal solution. Such problems typically require the satisfaction of some constraints. Some examples of optimization applications include the pricing and sale of passenger seats by airlines, the scheduling of manufacturing operations, for example, sequencing of tasks to machines to meet some criteria, finding the shortest of all possible distance of paths through a large number of cities, minimization of some cost function under a set of constraints, and so on. One of the earliest examples of ANNs in solving optimization problems was the application of dynamic recurrent networks to the traveling salesman problem [24].

Pattern Recognition

ANNs generally are good at learning perceptive type of tasks such as pattern recognition: handwritten or printed characters, visual images of objects, speech recognition, and so on. Many researchers increasingly are publishing the successful applications of ANNs in the areas of image processing, speech recognition, handwritten character recognition, automatic target recognition, robotics, process control, etc..

Forecasting

Prediction is a common task in many fields. A consumer products company will want to know the growth in sales for a new product they plan to introduce. Meteorologists need to predict the weather. Banks want to predict the credit-worthiness of companies as a basis for granting loans. Airport management groups want to know customer arrivals at busy airports, and power companies want to know customer demand for electric power in the future and so on. ANNs have been shown to be successful as predictive tools in a variety of ways: predicting that some event will occur or not, predicting the time at which an event will, or predicting the level of some event outcome. To predict with an acceptable level of accuracy, an ANN must be trained with a sizable set of examples of past pattern and future outcome pairs. The ANN must then be able to generalize and extrapolate from new patterns to predict associative outcomes. Some organization have developed sophisticated systems that require the training of hundreds or even thousands of ANNs on a weekly basis to predict stock market index movements as well as individual stock price behaviors.

Data Compression

Some ANNs are used to learn to compute a mapping that is a reduction of the input pattern space dimension to perform a sort of data compression. Patterns are transformed from n-dimensional space to m-dimensional space where $m < n$ by the assignments of codes to groups of similar patterns. The m-dimensional code words serve as prototypical patterns for whole clusters or groupings of similar patterns in n-dimensional space. Consequently, much shorter length patterns can be dealt with, thereby reducing the amount of transmission bandwidth required in data transmission applications and memory storage requirements when storing groups of patterns. Data compression is particularly important in applications where large amounts of data are being collected and processed as in the case of satellite image data processing.

Diagnosis

Diagnosis is a common ANN application for many fields: medicine, engineering, and manufacturing. This problem is essentially one of classification. It requires the correct association between input patterns that represent some form of symptom, or abnormal behavior, with the corresponding disease or equipment fault or other type of malfunction. Diagnosing complex systems, including ill people, is a practical expert system application. It is also a viable ANN application, and many diverse applications of diagnosis using some form of ANN architecture have been published in the literature.

3.1.2 History of Neural Networks Research

From the mid 1940s to 1960, in order to solve the basic principles for intelligent information processing, interdisciplinary research on the brain and computers was attempted. McCulloch and Pitts [35] published their important paper describing the properties of a simple two-state binary threshold type of neuron that has both excitatory and inhibitory inputs. Through proper choice of threshold levels, these units can be shown to perform any of the finite basic Boolean logic functions (inclusive OR, AND, NOT, and so on). Networks of these units were then thought to be representative models of the brain. They made a neuron model representing a basic component of the brain and showed its versatility as a logical operation system. Another early result was published by Hebb [21], who was one of the first to suggest a plausible process for neural learning. Even today, many of the learning models used by researchers are some form of *Hebbian* learning. Hebb had also reasoned that many distributed cell assemblies were used to represent knowledge, one of the first suggestions of the *connectionist* architecture. Rochester and colleagues [46] reported the computer simulation of ANNs at the Dartmouth summer conference now recognized as Artificial Intelligence (AI). In performing simulations of Herb's model, this group discovered that some changes were essential. They generalized the model to include inhibition as well, so that active cells could inhibit others from becoming active, and also introduced normalization of weights to prevent unbounded growth in some synapse weights. The concept of *fatigue* was introduced so that firing cells were less likely to fire in the immediate future. Although their work was not too conclusive, it was important as a forerunner of many simulation studies.

By the early 1960s, the perceptron received considerable excitement when it was introduced because of its conceptual simplicity. Some of them were Rosenblatt's perceptron [47] for the specific guidelines for learning systems by the proof of perceptron convergence theorem, and Widrow and Hoff's Adaline (adaptive linear neural element) [57] and Steinbuch's Learning Matrix [53]. The difference between the perceptron and the the Adaline lies in the training procedure. However in 1969, Minsky and Papert [37] introduced a rigorous analysis of the perceptron; they proved many properties and pointed out limitations of several models.

In the 1970s, Grossberg developed Adaptive Resonance Theory (ART) [17] based on biological and psychological evidence and proposed several architectures of nonlinear dynamic systems with novel characteristics such that the dynamics of the network were modeled by first order differentiable equations. These are self-organizing neural implementations of pattern clustering algorithms [9]. Von der Malsburg was perhaps the first to demonstrate self-organization using competitive learning through computer simulation [54]. Also, Albus developed an adaptive Cerebellar Model Articulation Controller (CMAC) which is a distributed table-look-up system based on models of human memory [2]. Anderson *et al.* [4] proposed the brain-state-in-a-box (BSB) model consisting of a simple associative network coupled to a nonlinear dynamics. Werbos [56] originally developed a back-propagation algorithm, and its first application was for estimating a dynamic model to predict nationalism and social communications. However, Werbos's work remained almost unknown in the scientific community [20].

In the 1980s, Hopfield introduced a recurrent-type neural network with symmetric synaptic called ***Hopfield networks*** consisting of a set of first-order differentiable nonlinear equations that minimize a certain energy function [23], and his approach was based on the Hebbian learning law [21]. Cohen and Grossberg established a general principle for designing a content addressable memory (CAM), which is known as the continuous-time version of the Hopfield network with a distinctive feature of an attractor neural network [10]. Furthermore, Kosko extended some of the ideas of Hopfield and Grossberg to develop his adaptive *Bidirectional Associative Memory* (BAM) [28] employing differential as well as Hebbian and competitive learning laws. Ackley *et al.* [1] and Hinton *et al.* [22] exploited the *Boltzmann machine* which is rooted in a *simulated annealing* process proposed by Kirkpatrick *et al.* [26]. The Boltzmann machine is trained by a two-phase Hebbian-based technique.

The neocognitron developed by Japanese researcher Kunihiko Fukushima [16, 15, 14] is a hierarchical feedforward network that learns through either supervised or unsupervised methods. The networks are modeled after biological visual neural systems. Fukushima and his colleagues have published results showing the neocognitron to be capable of hand written character recognition, independent of scale, position and some deformation in the characters. One version of the system is capable of identifying multiple characters using a feedback path.

One of the most important developments of recent neural network research is the discovery of a learning algorithm to adjust the weights in multilayer feedforward networks (also referred to as multilayer perceptrons). This algorithm is known as ***(error) back propagation*** since the weights are adjusted from the output layer backwards layer-by-layer to reduce the output errors. The method was discovered at different times by Werbos [56], Parker [42], and Rumelhart *et al.* [48]. This development opened the way for more general ANN computing by overcoming the limitations suffered by single-layer perceptrons. The backpropagation algorithm has emerged as the most popular one for a learning algorithm, and the two-volume book, *Parallel Distributed Processing* (PDP) was published by Rumelhart and McClelland [49]. This work gave a strong impulse to the neural networks research and revitalized the subsequent research in this field. This research in the 1980's triggered the present boom in the neural networks society. Also in 1988 a procedure of multi-layered feedforward networks was described by Broomhead *et al.* [8] using *Radial Basis Functions* (RBF), which has lead the design of neural networks to an area in numerical analysis and also linear adaptive filters. The research on RBF was further enriched based on the Tikhonov's regularization theory by Poggio in 1990 *et al.* [45].

3.2 Modeling of A Neuron

A biological neuron or nerve cell is believed to be the basic unit used for computation in the brain. The human apparently has something between 10^{10} and 10^{11} neurons, perhaps more. Somehow these cells are able to cooperate in an effective way.

3.2.1 Structure of Spinal Motor Neuron

Most of biological neurons in the literature for the artificial neural networks are modeled after the *spinal motor neuron*. Neurons possess tree-like structures called *dendrites* which receive incoming signals from other neurons across junctions called *synapses*. Some neurons communicate with only a few nearby ones, whereas others make contact with thousands. A simplified sketch of a biological neuron (spinal motor) is illustrated in Figure 8 and its functions are roughly explained as follows:

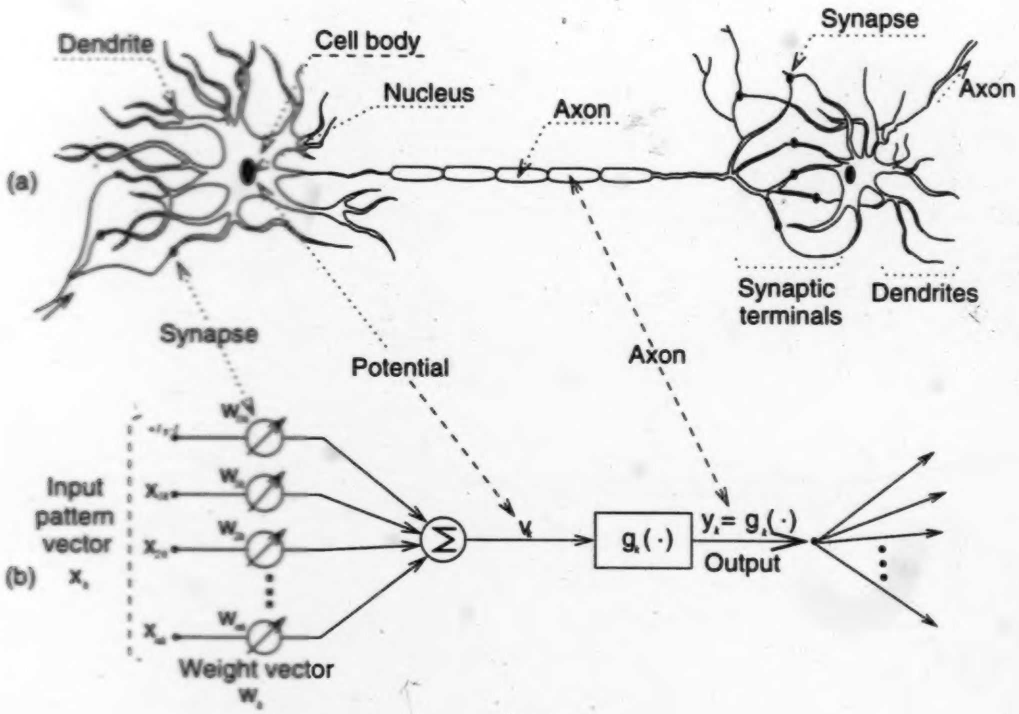

Dendrite
Cell body
Nucleus
Axon
Synapse
Axon
(a)
Synaptic terminals
Dendrites
Synapse
Potential
Axon
w_{0s}
Input pattern vector
x_s
(b)
x_{1s}
w_{1s}
x_{2s}
w_{2s}
x_{ms}
w_{ms}
Weight vector
w_s
Σ
v_k
$g_k(\cdot)$
$y_k = g_k(\cdot)$
Output

Figure 8: Comparative View of Biological and Artificial Neuron

- *Dendrites:* One end of the cell, the input end, has a number of fine processes called dendrites because of their resemblance to a tree (dendro- is a Greek root meaning tree). The variability in shape and size reflects the analog information processing that neurons perform.

- *Cell Body or Soma:* The cell body is referred to as the soma.

- *Axon:* Most neurons have a long, thin process, the axon, that leaves the cell body and may run for meters. The axon is the transmission line of the neuron. When axons reach their final destination they branch again in *terminal arborization* (arbor is Latin for tree).

- *Synapse:* At the ends of the axonal branches are complex, highly specialized structures called synapses. In the standard picture of the neuron, dendrites receive input from other cells, the soma and dendrites process and integrate the inputs, and information is transmitted along the axon to the synapses, whose outputs provide input to other neurons or to effector organs. Synapses allow one cell to influence the activity of others. Present dogma in neural network theory believes that synapses vary in strength, and that these strengths, that is, the detailed interactions among many neurons, are the key to the nature of the computation that neural networks perform.

- *Nucleus:* The nucleus and surrounding machinery have the job of sending nutrients, enzymes, and construction material down the axon to the rest of cell.

We can say that the neuron is a very busy place.

3.2.2 How Neurons Work

How neurons interact remains largely mysterious, and complexities of different neurons vary greatly. Generally speaking, a neuron sends its output to other neurons via its axon. An axon carries information through a series of action potentials, or wave of current, that depend on the neuron's voltage potential. More precisely, the membrane generates the action potential and propagates down the axon and its branches, where axonal insulators restore and amplify the signal as it propagates, until it arrives at a synaptic junction. This process is often modeled as a propagation rule represented by summation value $v = \sum_i^n$ in an artificial neuron of Figure 8.

A neuron collects signals at its synapses by summing all excitatory and inhibitory influences acting upon it. If the excitatory influences are dominant, then the neuron fires and sends this message to other neurons via the outgoing synapses. In this sense, the neuron function (activation function) can be modeled as a simple *threshold function*. Since the advanced knowledge on nervous systems is neither enough nor certain, it is almost impossible to exactly define the neuron functionalities and connection structures merely from a biological perspective. Accordingly, the selection of these activation functions generally depends on the applications the designed neural models are for, and artificial neuron models are loosely tied to the biological neuron abilities as Figure 8. Nevertheless, it is known that they have the potential to offer a truly revolutionary technology for modern information processing.

3.3 Artificial Models of the Neuron

Since neural networks are built up of interconnected model neurons, it is important to know what these neurons are supposed to do. From the discussion of elementary neurophysiology, it is clear that a real neuron is immensely complex, and it can be modeled at many levels of detail. If we tried to put into a model everything we know, or even a small fraction of what we know about a real neuron, it would be impossible to work with current computer technology. Therefore, we must use a model that is adequate for what we want to use it for. Choosing the level of detail to put in a model is something of an aesthetic judgment. A neuron model should be simple enough to understand and rich enough to give behavior that is interesting and significant. If all goes well, the result will predict things that might be seen in a real organism. If a practical application is in mind, it will be able to perform satisfactorily the function required of it. Many neuron models are used in the neural network literature. Here, three of them used currently are presented.

3.3.1 McCulloch-Pitts Neuron

McCulloch-Pitts neuron was first proposed by Warren McCulloch and Walter Pitts in 1943 [35]. McCulloch and Pitts made perhaps the first attempt to understand what the nervous system might actually be doing, given neural computing elements that were abstractions of the physiological properties of neurons and their connections. These investigators made a number of assumptions governing the operation of neurons that define what has become known as the *McCulloch-Pitts neuron*. In this model, the output of a neuron takes on the value of 1 if the total internal activity level of that neuron is negative and 0 otherwise. Therefore, the neuron is performing what is called *threshold logic* or *all-or-none*. Each neuron has a fixed threshold and receives inputs from excitatory synapses, all of which have identical weights. The neuron can also receive inputs from inhibitory synapses. If the inhibitory synapse is active, the neuron cannot become active. It is a binary device as shown in Figure 9

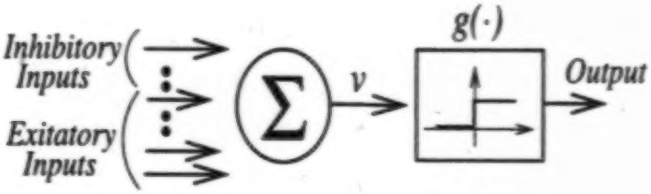


Figure 9: McCulloch-Pitts Neuron

The central result of the 1943 paper is that any finite logical expression can be realized by a network of McCulloch-Pitts neurons. This was exciting, since it showed that simple elements connected in a network could have immense computational power. It suggested that the brain was potentially a powerful logic and computational device. Does McCulloch-Pitts neurons make correct approximations to real neurons? Although it is occasionally suggested that they are adequate brain models and useful approximations to neurophysiology, this is not correct. But the work of McCulloch and Pitts gave rise to a tradition that views the brain as a kind of noisy processor doing logical and symbolic operations, much as a digital computer does.

3.3.2 Integration Neuron

Based on the neurophysiology of the horseshoe crab *Limulus*, Bruce Knight [27] analyzed a simple integration model of a single neuron. It is sometimes called the *integrate-and-fire* model of the neuron, and its close relative the *forgetful integrate-and-fire* are still used as simple neural elements for some physiologic applications. In Figure 10, imagine a noise-free neuron containing an internal variable $v(t)$, which might correspond to membrane potential. A stimulus $x(t)$ might correspond to ionic current.

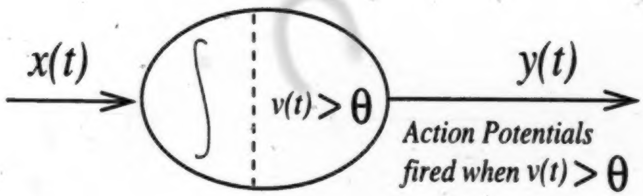


Figure 10: A integrate-and-fire neuron

Then,

$$\frac{dv}{dt} = x(t), \qquad (x(t) > 0).$$

When $v(t)$ reaches a criterion threshold θ, a nerve impulse is fired. The system resets $v(t)$ to its starting value after the spike is fired. Suppose a spike was fired at time t_1 and the time now is t. Then,

$$v(t) = \int_{t_1}^{t} x(\tau)d\tau.$$

A more complex version of the integration model assumes a decay of membrane potential due perhaps to membrane leakage. This assumption leads to a differential equation relating stimulus magnitude $x(t)$ and internal variable $v(t)$ with a decay term γ as follows:

$$\frac{dv(t)}{dt} = -\gamma \cdot v(t) + x(t)$$

The solution is now a little more complex, and the above expression can be integrated. More detailed relations about instantaneous firing frequency, stimulus magnitude $x(t)$ and decay factor γ is shown in [18], [19], [25].

3.3.3 Generic Connectionist Neuron

Although many neuron models are used in the neural networks architecture including the McCulloch-Pitts neuron, and integration neuron, the generic connectionist neuron is mainly considered here. This neuron has a two-stage process. In the first stage, inputs from the synapses are added together, with individual synaptic contributions combining independently and adding algebraically, giving a resultant activity level. In the second stage, This activity level is used to generate the final output activity of the model neuron by using the activity level as the input to a nonlinear function relating activity level (membrane potential) to output value (average output firing rate). Figure 11 shows the basic architecture.

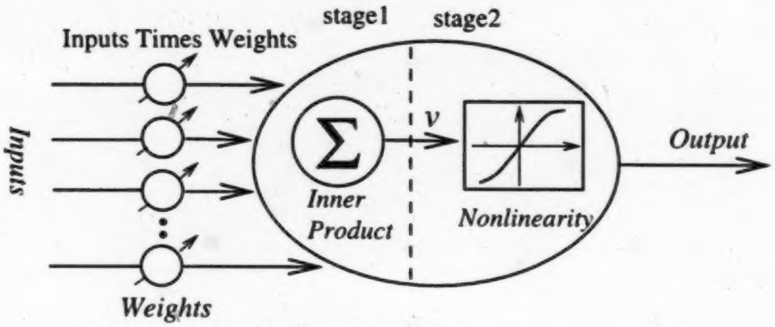


Figure 11: Generic Connectionist Neuron

Each artificial neuron model building up an artificial neural network simulates a biological neuron such that the basic artificial processing unit may be considered to have three components as follows:

1. *Weighted Summer* to sum the input signals weighted by the respective synapses of the neuron, whose operations constitute a linear combiner.

2. *Activation Function* to limit, or squash, the magnitude the output of a neuron. This neuron output function often uses one of the following: the two valued function of 1 and 0 using a threshold that is non-differentiable, a sigmoid function that is a continuous and differentiable nonlinear function, a Gaussian function, a ramp, a step, a linear with saturation, etc.. The expressions below show some typical activation functions:

$$\text{Threshold Function:} \qquad g(v) = \begin{cases} 0 & \text{if } v \geq 0 \\ 1 & \text{if } v < 0 \end{cases}$$

$$\text{Sigmod Function:} \qquad g(v) = \frac{1}{1 + e^{-av}}$$

$$\text{Gaussian Function:} \qquad g(v) = e^{(-v^2/\sigma^2)}$$

$$\begin{aligned} \text{Hyperbolic Tangent Function:} \qquad g(v) &= b \cdot tanh(\frac{a}{2} \cdot v) \\ &= b \cdot \left(\frac{1 - e^{-av}}{1 + e^{-av}}\right) \end{aligned}$$

A $tanh(v)$ is often used instead of the conventional sigmoid function since the output value of the sigmoid function is positive, whereas that of the $tanh(v)$ function is both positive and negative.

The choice of activation function is not clear, but generally they depend on the application based on the fact that neurons located in different parts in the nervous system have different characteristics. For example, the neurons of the ocular motor system have a *sigmoid characteristic*, while those located in the visual area have a *Gaussian characteristic* [5]. There are other useful function types such as *logarithmic* and *exponential* [11] although their biological basis has not been established.

3. *Weight, Synapse or Connecting Link* to be characterized by a strength of its own. It is universally assumed that the connection strengths are single numbers; the more complex temporal behavior and nonlinear interactions known to exist in real synapses are ignored in the name of simplicity. Almost all neuron models assume linear addition of synaptic interactions as the first stage in the information processing.

- *Bias or Threshold* inputs to have the effects of increasing (called bias) or lowering (called threshold) the net input of the activation function respectively. Generally, their values are selected as +1 and −1 respectively. Biased networks can more easily represent the relationship between the inputs and outputs than networks without biases. For example, when all inputs to the network are zero, a neuron without a bias will always have a zero output. A neuron with a bias, however, can learn transfer functions that have a non-zero output under the same conditions by feeding an appropriate value for the bias. (However, it is difficult to determine the appropriate value for the bias. **In this research, a new neuron model is proposed to make an automatic decision on bias/threshold value of sign and magnitude.**)

Consequently, the mathematical output of the neuron unit is expressed as follows:

$$\begin{aligned} y &= g(v) \\ &= g(\sum_{i=1}^{n} w_i \cdot x_i \pm \theta_i) \end{aligned}$$

3.4 Neural Networks Classification

In the biological brain, a tremendous number of neurons are interconnected to form the network and perform advanced intelligent activities. For example, the human brain is estimated to have about 10^{11} neurons, and for each neuron, the number of connections can be up to 10^3 to 10^4, resulting in up to 10^{14} to 10^{15} interconnections in the neural system of the human brain. In neural networks, therefore, connections are important. There can be many different ways to connect artificial neurons or processing units in large networks. These different patterns of interconnections between the neurons are called *architectures.* Network architecture, therefore, consists of how many neurons each layer has, the layers a network has, each layer's transfer function, and how the layers are connected to each other and to networks inputs. The best neural architecture to use depends on the type of problem to be represented by the network. Large networks may be able to perform complex tasks which would be impossible with individual neurons. For that reason, **the development of various problem-specific architectures of neuron-like networks are required and the derivation of learning algorithms associated with these architectures must be performed.**

ANNs can be classified depending on different basis such as the *learning paradigm*, the *connection architecture* and general *area of application.* For example, based on the learning paradigm, one can make the network classification of supervised/unsupervised and reinforced learning networks or based on the application type, network classification of prediction, pattern recognition, classification and associative memory networks. In this section, the classification of network type is based on connection architecture since the behavior of the network depends largely on the interactions between the neurons and synaptic weights. For this, two classes have been roughly defined as feedforward and recurrent neural networks.

3.4.1 Feedforward Neural Networks

- **Single Layer Feedforward Networks:** Networks with a single layer of computational neurons that process input signals in a feedforward direction include ADALINE (Adaptive Linear Neural Element), Perceptron, AM (Associative Memories), LVQ (Learning Vector Quantization), and SOFM (Self-Organizing Feature Map).
- **Multi Layer Feedforward Networks:** Networks with two or more layers of connections with weights that process the inputs in a forward direction, with which data from neurons of a lower layer are propagated to neurons of an upper layer include MADALINE (Multiple Adaptive Linear Element), RBF (Radial Basis Function), RCE (Reduced Coulomb Energy), CCN (Cascade Correlation Networks), GRNN (Generalized Regression Neural Networks), MLFF with BP (Multilayer Feedforward Backpropagation), and Neocognitron.

Feedforward networks can only perform **static processing**; i.e., all free parameters have fixed values. A typical feedforward network consists of an input layer, one or more middle layers and an output layer. The middle layer(s) is called *hidden* because they only receive internal inputs from other processing units and produce internal outputs to other processing units. Accordingly, they are hidden from the outside world. Activation signals of neurons in one layer are transmitted to the next layer through a set of links which either attenuate or amplify the signal based on the corresponding weight.

The learning phase of a layered neuron is the process where all its weights are adjusted according to a specified learning rule in order to minimize a specified *cost function* (objective function). A commonly used cost function E is the *mean square error* between the actual neural network outputs and the specified targets for a set of N training patterns in case of off-line (or batch mode) method. *The weight updating problem is to find a set of weights that minimizes the predefined cost function.* For example of fixed connection weight, the outputs of the trained static network are supposed to respond to a given input pattern to meet the defined objectives. This is often achieved by gradient descent method. A variation of the descent method suitable for the multilayer neural network structure is commonly used and called **Error Backpropagation** algorithm. **It is the most widely used method for training.**

Multilayer neural networks can be used for both classification and system identification. In classification, the neural network is trained to be the discriminant function based on a collection of correctly classified examples. The success of the classifier depends on its ability to correctly classify previously unseen patterns. Multilayer perceptrons are known to have superior generalization and noise rejection capability which makes them well-suited for the task. A distinct advantage in layered perceptron classifiers is that their complexity is determined by the number of neurons, the nature of the interconnections, and the type of nonlinear threshold used. *This leaves the question of how to determine the optimal neural network architecture for a given problem.* Conventional wisdom is that for good generalization ability, one has to build into the neural network as much knowledge about the problem as possible while limiting the number of interconnections. There are a couple of theorems and techniques related to the architecture itself that are worth noting. **A single hidden layer is always enough to approximate any continuous function although how many hidden units are required is not generally known.** Therefore, particular diligence must be applied when choosing *hidden* layer configurations for a multilayer perceptron. An architecture with too few hidden neurons results in a network that is unable to distinguish patterns; an architecture with too many hidden neurons can result in a network that is simply memorizing the training set patterns. It will be unable to make generalizations required to correctly process the test data upon presentation. An approach is to incrementally modify the architecture to suit a particular task. One can remove non-useful connections during training. After considerable training, the dead neurons can be dropped from the architecture. Rather than pruning a larger network, another approach is to start with a small network that is gradually enlarged. Both methods have given encouraging results but more studies are needed before their applications can be justified.

3.4.2 Recurrent Neural Networks

- **Recurrent (Feedback) Neural Networks:** Networks that have feedback connections which propagate the outputs of some neurons back to the inputs of other neurons sometimes through time-delayed elements (including self-feedback and lateral connections) to perform repeated computations on the signals. These networks are more suitable for *dynamic* models, which include RNN (Recurrent Neural Networks), BAM (Bidirectional Associative Memory), ART (Adaptive Resonant Theory), BSB (Brain-State-in-a-Box), Boltzmann Machine, Cauchy Machine, and Hopfield.

There are ways to provide the mapping network with dynamic properties that make it responsive to time-varying signals. In short, **for a neural network to be dynamic, it must be given memory** [12]. One way in which this requirement can be accomplished is to introduce *time-delay* into the synaptic structure of the network and to adjust their values during the learning phase. The use of time delays in neural networks is neurobiologically motivated, since it is well known that signal delays are omnipresent in the brain and play an important role in neurobiological information processing [7, 36]. The time delay neural network (TDNN) was first described by [29, 55]. **Another way in which a neural network can perform dynamic behavior is to make it recurrent**, i.e., to build feedback into its design. Recurrent neural networks (RNNs) are constructed in a recurrent (feedback) manner by connecting the output of one or more neurons to the inputs of one or more neurons in the same or preceding layers. These feedback architectures may have lateral connections among neurons of the same layer including self-feedback connections on the same neuron as well. Incorporating feedback connections into feedforward networks results in significant changes in the operation and learning processes of the networks as compared to their static feedforward counterparts.

RNNs exhibit dynamic behavior unlike static feedforward networks. Therefore, it can be said that the feedback signals are responsible for the dynamic behavior of the recurrent network systems. They can perform mappings that are functions of time. As a result, they are capable of performing more complex computations than static feedforward networks. For some applications, a recurrent network may need an arbitrarily large number of hidden-layer nodes in order to realize the same performance levels. The behavior of RNNs can be better understood with a knowledge of nonlinear dynamic phenomena, such as turbulent fluid flows or nonlinear control systems. Such systems also have coupled interdependencies and feedback paths. Their behaviors or dynamics are governed by sets of coupled nonlinear differential (or can be modeled by difference) equations. Likewise, the dynamics of RNNs can be completely described by set of first order nonlinear differential (difference) equations of the form:

Nonlinear differential equation:

$$\frac{dy_i}{dt} = g_i(\mathbf{w}, \mathbf{x}, \mathbf{y}(t)) \qquad i = 1, 2, \ldots, n$$

Nonlinear difference equation:

$$y_i(k+1) = g_i(\mathbf{w}, \mathbf{x}, \mathbf{y}(k)) \qquad i = 1, 2, \ldots, n$$

where $\mathbf{y}(t)$ is the state vector while $y_i(k)$ is the output of the ith neuron at time k, $\mathbf{w}$ the matrix of synaptic connection strengths, $\mathbf{x}$ the external input vector and g_i is a nonlinear differential function. For systems with recurrence or feedback, questions of stability become relevant to the parameters $\mathbf{x}$, $\mathbf{w}$ and the initial conditions of $\mathbf{x}(0)$, $\mathbf{w}(0)$ and $\mathbf{y}(0)$. The behavior is involved in one of four ways:

- convergence to a stable attractor point,
- settle to cyclical oscillations,
- tend toward quasiperiodic behavior (oscillations at multiple frequencies), and
- exhibit a form of chaotic wandering behavior.

We will be concerned primarily with the stable convergence case where the network converges to a single attractor point or performs some desired mapping on the input vector. Otherwise, some conditions can be stated to insure a sufficiently stable behavior. The behavior of RNNs have been studied and described by [22, 44, 43, 59, 62, 58, 13] among others, and RNNs have been used in a number of interesting applications including control, optimization, forecasting, and pattern classification for speech recognition.

3.5 Neural Network Training Algorithm

A training or learning of neural networks are associated with any change in the neural connections as represented by the synaptic weights. (Training does not imply a change in the structure of networks since structural learning is a separate issue.) Training can be regarded as a parametric adaptation algorithm. A process of automatic weights adjustment is generally called *learning* in the ANNs literature and *adaptation* in the control literature. Neural networks will either have fixed weights or adaptable weights. Networks with adaptable weights use learning algorithms to adjust the values of the synaptic strengths. Network with fixed weights is to use fixed (constant) weights which are determined explicitly from the well defined problem a priori. If the correct weights values are not known a priori, adaptable weights by the training algorithms are essential.

There are two types of training (or learning) algorithms: *supervised* and *unsupervised* depending on the use of an external teaching signal. Supervised learning occurs when the network is supplied with both the input values and the correct output value (or external teaching signal), and this learning algorithm adjusts the networks's weights to minimize the error between the network output and the external teaching output. Unsupervised learning occurs when the network is only provided with input values, and this learning method adjusts the weights based solely on the input values and current network output. The training algorithms can also be classified into *off-line* (batch mode) or *on-line* methods. Off-line and on-line training techniques can be applied to a *time-invariant* system, and on-line training to a *time-varying* system in the context of control applications. A variety of learning algorithms has been proposed such as backpropagation (BP), competitive learning, and Boltzman learning. BP, especially, is a systematic procedure to train multilayer ANNs and a form of gradient descent optimization.

3.5.1 Supervised Training: Backpropagation Algorithm

The error backpropagation (BP) method is typical of one of the supervised learning techniques, that was developed by Werbos [56], and rediscovered and popularized in an entirely different context by Rumelhart *et al.* [49]. In fact, *the popularity of the Error BP algorithm made feedforward neural networks almost synonymous with supervised learning.* This training algorithm is an iterative gradient method designed to minimize the mean square error between the actual output of a feedforward and feedback (recurrent) net and the desired output. Above all, **BP solves the problem** of hidden layer learning for multilayer networks, which is why [49]'s contribution is widely recognized. It requires continuous differentiable nonlinearities of activation functions [49, 20]. During training, weights are updated to move the network closer to the given network output target in the following manner:

- Each output unit's computed activation is compared with its target value to determine the associated error for a particular pattern.
- An error-dependent factor is calculated. This factor is propagated back to the previous layer of neurons. Later, it is used to update the weights leading to the output layer neurons.
- Similarly, error factors are propagated recursively back through each layer of (hidden) neurons until the input layer is reached. Input neurons are not changed.
- All weights are adjusted simultaneously at the end of each epoch based on the error factors at each layer.

In the BP algorithm, the learning coefficient should be defined for the *rate of learning*, and *momentum value* adds a tendency for weights to change in the direction they have been changing. The BP algorithm can be seen as a gradient algorithm applied to a nonlinear optimization problem. Briefly, the BP algorithm solves missing information problems as follows:

- Inputs to the hidden layers are taken as the inputs to the first layers propagated forward through the network.
- The effective reference signals for the hidden layers are obtained by backpropagation of the error through the network. This is achieved by taking the partial derivative of the squared error against the weight parameters.

Overall, supervised learning is an optimization process aimed at minimization of the error cost function with respect to weights. Due to the nonlinearity of neural

networks, the (error) cost function with respect to weights possesses many **local minima** resulting in suboptimal solutions when gradient (a kind of hill climbing) methods like standard BP are applied. **Local minima problem should be definitely overcome in the context of training algorithm**. In this supervised learning scheme, two alternative approaches exist, namely, off-line training and on-line training. In the off-line training scheme, a set of input-output samples of the teachers are given in advance during a training period, while in the on-line training scheme, a sample of each input-output of the teacher is observed at every sampling instance and the training algorithm updates its weights only once during a sampling period. Therefore, a time-varying system can be analyzed effectively based on on-line training scheme.

3.5.2 Unsupervised Training

Unsupervised training attempts to cluster similar patterns together without using training data. In theory, the patterns are classified based on similarities between input vectors - there are no targets. **Network weights are updated so that similar input vectors are assigned to the same output cluster**.

Classification decisions are made by assigning input vectors to the cluster that features the most similar exemplar vector. An exemplar vector is the role model or representative of a class of vectors. It is used as a basis of comparison prior to admitting another vector to the class. Some networks rigidly maintain their exemplars, while others update exemplars to reflect the most recent addition. This is done either by making the new entry to a class the exemplar vector or through some variation of a weighted average of all of the vectors in a cluster.

For example, a feedforward neural network may be trained without a teacher according to some learning rule which imposes a certain condition on its output. Feedforward neural networks trained without a teacher may measure the correlation of the input data, identify certain features, or perform principal component analysis. The unsupervised training of feedforward neural networks was extensively studied during the past decade [32, 31, 30, 33, 38, 39, 40, 41, 60, 50, 51, 52, 6]. A common ingredient in almost all these studies is the Hebbian learning rule, which is a biologically inspired scheme and has strongly influenced unsupervised learning [21, 3].

3.6 Remarks on ANNs

Although the actual working mechanism of the human brain has not been totally understood yet, ANNs are capable of simulating some functionalities of the human brain by using a simplified model of the human neural system. Accordingly, the development of artificial neural systems requires cross-disciplinary research, covering neural science, biology, psychology, computer science, computer engineering, signal and image processing, mathematics, physics, optics, and VLSI electronics technology. The field of neural networks has so far attracted researchers from different backgrounds. ANNs provide a unified common language for many diverse disciplines. Therefore, the ultimate goal is to attain a unified study on algorithms, architectures and applications for neural networks. If fundamental ANNs system theory is established, it would serve as a foundation for future research advances and long-term applications.

Bibliography

[1] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, vol.9:pp.147–169, 1985.

[2] J.S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *J. Dynamic Syst. Meas. Contr.*, pages 220–227, 1975.

[3] J.A. Anderson. Cognitive and psychological computation with neural models. *IEEE Transactions on Systems, Man and Cybernetics*, vol.13:pp.799–815, 1983.

[4] J.A. Anderson, J.W. Silverstein, S.A. Ritz, and R.S. Jones. Distintive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review*, vol.84:pp.413–451, 1977.

[5] D.H. Ballard. *Cortical connections and parallel processing: Structure and function.* MIT Press, Cambridge, MA, 1988. In Vision, Brain, and Cooperative Computation, M. Arbib and Hamson (Eds.).

[6] W. Banzhaf and H. Haken. Learning in a competitive network. *Neural Networks*, vol.3:pp.423–435, 1990.

[7] V. Braitenberg. Two views of the cerebral cortex. In G. Palm and eds. A. Aertsen, editors, *Brain Theory*, pages 81–96. Springer-Verlag, New York, 1986.

[8] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, vol.2:pp.321–355, 1988.

[9] G.A. Carpenter and Eds. S. Grossberg. *Pattern Recognition by Self-Organizing Neural Network.* MIT Press, Cambridge, MA, 1991.

[10] M.A. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:pp.815–826, 1983.

[11] W.J. Daunicht. Defanet-a deterministic approach to function approximation by neural networks. *IEEE Int. Joint. Conf. on Neural Networks, IJCNN'90*, pages 161–164, 1990.

[12] J.L. Elman. Finding structure in time. *Cognitive Science*, vol.14:pp.179–211, 1990.

[13] J.L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, vol.7:pp.195–225, 1991.

[14] T. Fukuda and T. Shibata. Research trends in neuromorphic control. *J. Robotics Mechatron*, vol.2(no.4):pp.4–18, 1991.

[15] K. Fukushima, , S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybertetics*, SMC-13:pp.826–834, 1983.

[16] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformation and shifts in posion. *Pattern Recognition*, vol.15(no.6):pp.455–469, 1982.

[17] S. Grossberg. Adaptive pattern classification and universal recording ii: Feedback, expectation, olfaction, and illusions. *Biol. Cybern.*, vol.23:pp.187–202, 1976.

[18] H.K Hartline and C.H. Graham. Nerve impulses from single receptors in the eye. *Journal of Cellular and Comparative Physiology*, vol.1:pp.1049–1066, 1932.

[19] H.K. Hartline and F. Ratliff. Spatial summation of inhibitory influences in the eye of limulus and mutual interaction of receptor units. *Journal of General Physiology*, vol.41, 1958.

[20] Simon Haykin. *Neural Networks : A comprehensive Foundation.* Macmillan College Publishing Company Inc., New York, 1994.

[21] D.O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[22] G.E. Hinton and T.J. Senowski. *Learning and relearning in Boltzman machines.* MIT Press, Cambridge, MA, 1986. In Parallel Distributed Processing: Explorations in Microstructure of Cognition (D.E. Rumelhart and J.L. McClelland, eds.).

[23] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.*, vol.79:pp.2554–2558, 1982.

[24] J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, vol.52:pp.141–152, 1985.

[25] E.F. MacNichol Jr. Visual receptors as biological transducers. In R.G. Grenell, editor, *Molecular Structure and Functional Activity of Nerve Cells*, pages 34–53, Washington, DC, 1955. American Institute of Biological Sciences.

[26] S. Kirkpatrick, Jr. C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, vol.220:pp.671–680, 1983.

[27] B. Knight. Dynamics of encoding in a population of neurons. *Journal of General Physiology*, vol.59:pp.734–766, 1972.

[28] B. Kosko. Adaptive bidirectional associative memories. *Appl. Optics*, vol.26:pp.4947–4960, Dec. 1 1987.

[29] K.J. Lang and G.E. Hinton. The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University, 1988.

[30] R. Linker. From basic network principles to neural architecture: Emergence of orientation columns. *Proceedings of the National Academy of Sciences U.S.A.*, vol.83:pp.8779–88783, 1986.

[31] R. Linker. From basic network principles to neural architecture: Emergence of orientation-selective cells. *Proceedings of the National Academy of Sciences U.S.A.*, vol.83:pp.8390–8394, 1986.

[32] R. Linker. From basic network principles to neural architecture: Emergence of spatial-opponent cells. *Proceedings of the National Academy of Sciences U.S.A.*, vol.83:pp.7508–7512, 1986.

[33] R. Linker. Toward an organizing principle for a layered perceptual network. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 485–494, New York, 1988. American Institute of Physics.

[34] R.P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Mag.*, vol.78:pp.4–22, April 1987.

[35] W.S. McCulloch and W.H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophisics*, vol.5:pp.115–123, 1943.

[36] R. Miller. Representation of brief temporal patterns, hebbian synapses, and theleft-hemisphere dominance for phone recognition. *Psychobiology*, vol.15:pp.241–247, 1987.

[37] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969. Introduction, pp.1–20.

[38] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, vol.15:pp.267–273, 1982.

[39] E. Oja. *Subspace methods of pattern recognition*. Research Studies Press and J. Wiley, Letchworth, England, 1983.

[40] E. Oja. Nonlinear networks, principal components, and subspaces. *International Journal of Neural Systems*, vol.1:pp.61–68, 1989.

[41] E. Oja. Learning in nonlinear constrained hebbian networks network. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 385–390, North Holland, 1991. Elsevier Publishers.

[42] D.B. Parker. Learning logic. *Technical Report TR-47, Ceneter for Computational Research in Economics and Management Science, MIT*, 1985.

[43] B.A. Pearlmutter. Dynamic recurrent neural networks. Technical Report Report CMU-CS-88-191, School of Computer Science, Pittsburgh, PA, 1988.

[44] F.J. Pineda. Dynamics and architectures for neural computation. *Journal of Complexity*, vol.4:pp.216–45, 1988.

[45] T. Poggio and F. Girsi. Networks for approximation and learning. *Proc. IEEE*, vol.78(no:9):pp.1481–1497, Sept. 1990.

[46] N. Rochester, H. Holland, L.H. Haibt, and W.L. Duda. Tesst on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory*, IT-2:pp.80–93, 1956.

[47] F. Rosenblatt. The perceptron:a probabilistic model for information storage and organization in the brain. *Psychological Rev.*, vol.65:pp.368–408, 1958.

[48] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representation by back-propagating errors. *Nature (London)*, vol.323:pp.533–536, 1986.

[49] D.E.
Rumelhart and J.L. McClelland. *Parallel Distributed Processing:Explorations in the Microstructure of Cognition*, volume vol.1. MIT Press, Cambridge, MA, 1986.

[50] T.D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, vol.2:pp.459–473, 1989.

[51] T.D. Sanger. Optimality principle for unsupervised learning. In D.S. Touretzky, editor, *In Advances in Neural Information Processing Systems 1*, pages 11–19, San Mateo, California, 1989. Morgan Kaufmann.

[52] T.D. Sanger. Analysis of the two-dimensional receptive fields learnned by generalized hebbian algorithm in respose to random input. *Biological Cybernetics*, vol.63:pp.221–228, 1990.

[53] K. Steinbuch and V.A. Piske. Learning matrix and their applications. *IEEE Trans. Electron. Comput.*, EC-12:pp.846–862, 1963.

[54] C. von der Malsberg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, vol.14:pp.85–100, 1973.

[55] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. on Acoustics, Speech and Signal Processing*, ASSP-37:pp.328–339, 1989.

[56] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, Cambridge, MA, Aug. 1974.

[57] B. Widrow and M.E. Hoff Jr. Adaptive switching circuits. *IRE WESCON Covention Record*, pages 96–104, 1960.

[58] R.J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, vol.2:pp.490–501, 1990.

[59] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, vol.1:pp.270–80, 1989.

[60] A.L. Yuill, D.M. Kammen, and D.S. Cohen. Quadrature and the development of orientation selective cortical cells by hebb rules. *Biological Cybernetics*, vol.61:pp.183–194, 1989.

[61] M. Zak. An unpredictable-dynamics approach to neural intelligence. *IEEE Expert*, pages 4–10, 1991.

[62] D. Zipser. A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks. *Neural Computation*, vol.1:pp.552–8, 1989.

Chapter 4

Architecture Of Recurrent Neural Networks

4.1 Introduction

In the area of control and signal processing, a designer often tries to find a *model* capable of producing numerical values that match values measured from some physical system or process or plant. This kind of work is known as *system identification* [22],[21]. It is assumed that the model has a specific mathematical form expressed in terms of a set of unknown parameter values and one can use a parameter estimation method to find good parameter values. This estimation method requires the consecutive data pair of input and output values from a teacher (or the plant to be controlled).

In neural networks, the technique of determining a specific mathematical form and a topology for the mathematical form is called *neural network architecture*. The parameter (synaptic weight) estimation method is called *learning* or *training*. Since the use of a neural network allows a more general parameter identification setup, a nonlinear system can be identified using a neural network. Moreover, in a closed-loop operation, the neural network can be trained to balance the requirements of identification and control. Frequently, it is shown that the neural identifier is better than an extended Kalman filter in terms of robustness [17]. Incorporation of time-delay feedback into temporal dynamic models leads to the so-called *recurrent neural network* (RNN) [26],[35]. A multi-layer network may be made recurrent by introducing time-delay loops to the input, hidden, and/or output layers. Another way is to route delay connections from one layer to another - called *inter-layer* connections in this thesis. As a result of such a structural change, signals in the networks go through both time and space. RNN models can learn more than just static mapping between the input and output due to their capability to store information and attractor dynamics [9],[10],[27],[15]. Thus, RNNs have a capability of *dynamic mapping* and are better suited for dynamical systems than the feedforward networks. In this chapter, some architectural techniques for recurrent and feedforward ANNs are devised.

4.2 Dynamic Properties By Feedback and Memory

In a static model, all free parameters have fixed values. Therefore, a static structure maps an input vector onto an output vector. This form of static input-output mapping is well-suited for *pattern-recognition* applications like handwritten character recognition, where input and output vectors represent *spatial* patterns which are independent of time. The static structure may also be used when a nonlinear prediction is performed on a stationary time series, where its statics do not change with time,

However, dynamics of a fast time-varying nonlinear process is in general time-history dependent and cannot be modeled by static input-output maps. One knows that time is important in many of the cognitive-type tasks such as signal processing (vision, speech, etc.) and control applications encountered in practice. The question is how to represent time so that a time-varying form will be able to deal with time-varying signals. One solution to this problem is to provide the mapping network with dynamic properties which make it responsive to temporal sequences or time-varying signals. In order for a neural network to be *dynamic*, it must be given *memory* or a *time-delay unit* [6]. There are many ways in which this can be accomplished, and a number of interesting proposals have appeared in the literature [14], [25],[30],[31],[32],

[33],[34].

One way to do this is to introduce internal *time-delay* units into the synaptic weights of the network and to adjust their values during the training phase[24], [8]. This is called a time-delay neural network (TDNN), which was first described by [18], [32], and can be used for modeling nonlinear dynamics and reducing the number of inputs needed for the modeling task. Thus, a TDNN can cope with multi-dimensional processes[24]. By introducing more than one delay element at each unit any higher dynamical model can be achieved without increasing the number of inputs. A similar approach is reported in [13], where the authors propose different local ARMAX models being weighted by the associated basis function. The use of time delays in neural networks is neurobiologically motivated, since it is well known that signal delays are omnipresent in the brain and play an important role in neurobiological information processing[2], [3],[4],[23].

Another way to provide a neural network with dynamic behavior is to make it *recurrent*, that is, to build feedback into its design. External time-delay units can be used for the recurrent signals to embed more *temporal* behavior in the model, which is contrary to the *spatial* behavior prevalent in pattern recognition. A key advantage of recurrent networks lies in their ability to use information about past events for current computations. Thus they can provide time-dependent outputs for both time-dependent as well as time-independent inputs.

4.3 Jordan's RNN

Jordan[14] developed a network containing recurrent connections that were used to associate a static pattern (a *Plan*) with serially ordered output patterns (a sequence of *Actions*). The recurrent connections allow the network's hidden units to see their own previous output, so that the subsequent behavior can be shaped by previous responses. These recurrent connections are what give the network memory. Jordan's network played a useful role in motor control and robotics applications. The uniqueness of Jordan's model is that it dealt with time explicitly (as opposed to representing temporal information spatially in the TDNN [18],[32]). This is realized by adding recurrent links from output to input and from input to itself, and by providing a time-varying error function. This is shown in Figure 12. The architecture specified by the Jordan employs first-order connections between units. Connections from output to state units are one-for-one with a fixed weight of 1.0.

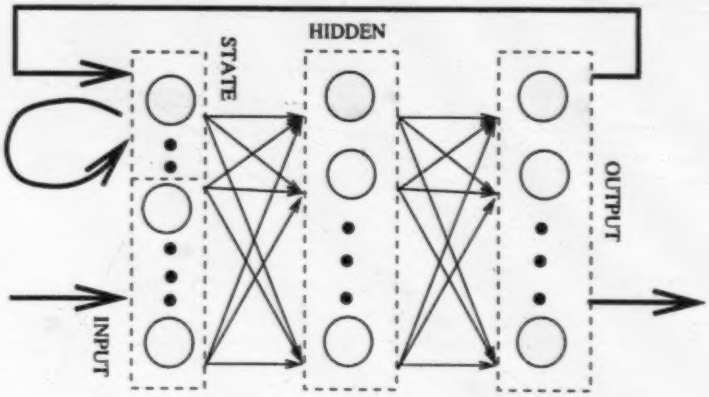


Figure 12: Jordan's Recurrent Neural Network

Learning method for Jordan's recurrent neural network is an extension of the backpropagation learning method. A general learning algorithm is that of Williams and Zipser [34]. Jordan's learning method is regarded as a special case of Williams and Zipser algorithms as shown in [16]. Jordan's RNNs have appeared in a variety of control applications.

4.4 Elman's RNN

Elman [5],[6] developed a Simple Recurrent Network (SRN) or Elman's RNN, which has a two-layer network with feedback in the first layer as shown in Figure 13.

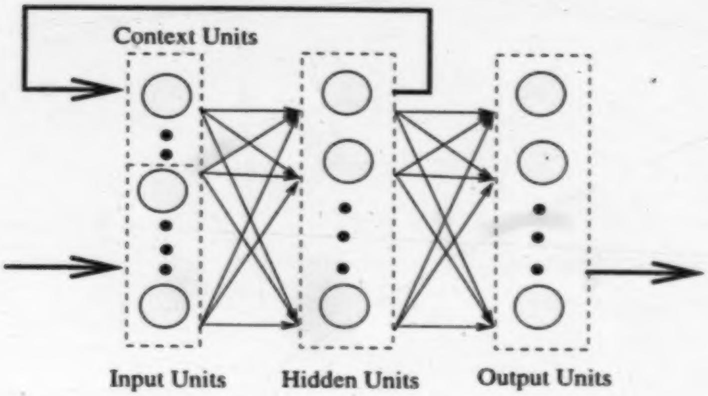


Figure 13: Elman's Recurrent Neural Network

The Elman network has neurons with a hyperbolic-tangent transfer function in its hidden layer, and neurons with linear transfer function in its output layer. This combination is noticeable in that two-layer networks with these transfer functions can approximate any function provided that a finite number of discontinuities are assumed and the hidden layers have enough neurons. Therefore, more hidden neurons are needed as the function being fit increases in complexity. It is noted that the Elman's RNN differs from conventional two-layer networks in that the first layer (relatively hidden layer) has a recurrent connection (fixed at 1.0) to the input level by additional units called *Context Units.* These units are also hidden in the sense that they interact exclusively with other nodes internal to the network, and not the outside world. The importance of internalizing the recurrent links is that the internal layers' activation is only indirectly constrained through learning, whereas the activation in the input and output layers is explicitly determined by the user.

The delay in this connection stores values from the previous time step, which can be used in the current step. Thus, even if two Elman networks, with the same weights and biases, are given identical inputs at a given time step, their outputs can be different due to different feedback states. Elman's network may be trained to respond to temporal patterns as well as spatial patterns. The learning method for Elman's RNN is also an extension of the backpropagation algorithm as a special case of Williams and Zipser [34] shown in [16]. Elman's RNNs have been often applied to the problem of pattern recognition (or symbolic sequence prediction).

Elman's RNN has been used by other authors [28],[29],[20], and may be useful in various time-related sequences such as language. However, despite a few pioneering efforts and a growing informal consensus on the strengths and weakness of the model, little work has been done to determine the effect of its numerous parameters and to evaluate the types of sequences it performs best. One of the many difficulties with neural architecture is that formal analysis and detailed methods on a dynamical neural model seldom exist for it. Hence, empirical exploration seems the only current way of understanding such a neural model. [12] is of the opinion that Elman's Simple Recurrent Networks may have problems when it is applied to difficult tasks such as language processing, and when used for more sophisticated models for a complete solution.

4.5 Proposed SERNN

This section proposes a new type of neural architecture called the *Supervision & Error Recurrent Neural Network* (SERNN), where supervision and error stand for teaching signal and residual error between actual plant and neural model respectively. A rough schematic diagram of the proposed recurrent neural net is shown in Figure 14. *The distinct features of the SERNN are that it introduces new feedback inputs from teaching signals and from mapping (identification) errors between teaching signals and neural outputs, and the feedforward inter-layer synaptic weights.* The new inputs create additional and meaningful synaptic weights for external teaching and error signals respectively.

The external supervision feedback is based on the intuitive assumption that more precise information on the system being mapped will be obtained from the actual system than from the output of a neural model. In all conventional recurrent architectures, the output feedback of the neural model has been performed, which may lead to a side-effect of undesirable output-feedback especially at the beginning of neural training with improper initial weight values. This kind of side-effect is defined as *continuous vicious cycle of poverty* in economics. Recurrent supervision is to remove this side effect. Many neural training algorithms have resulted in poor performance in terms of precision and speed in the learning phase. Therefore, supervision feedback imply the notion that the neural training is to reproduce the supervision signal by the fast, robust and accurate manner.

Conventional neural nets do not have meaningful bias/threshold value and reasonable interpretation for that on each neuron. This research develops the automatic way of selecting the bias/threshold value and presents its reasonable interpretation. This is carried out by introducing feedback of the training error (or identification error or mapping error). In a conventional neuron model the bias and threshold are added externally and have the effect of increasing and lowering the network input of the activation function respectively[8]. However, it is difficult for the ANN designer to determine their magnitude and sign since they depend on the particular system and to find the physical interpretation for different systems. In the new architecture proposed in this thesis, these values are determined as follows: *Input for threshold or bias takes values from an error signal for the supervised training* and the *synaptic weight is added for the recurrent error signals.* This modification allows the neuron model to make an automatic decision regarding sign and magnitude for threshold or bias value instead of manual selections of -1.0 and $+1.0$ by a designer. *Recurrent error is expected to make a direct contribution to fast tracking for an identification because it creates a new and meaningful input source for RNNs. This idea originated in this thesis from the assumption that the neurons (spinal motor, for example) of human brain obtains and keeps using a kind of correction (error in ANNs) signal from outside through sensors, and memory (experiences, that is, database) to make decisions under an unfamiliar environment. Finally, the brain reaches a confident (= stable in ANNs) status (= state in ANNs) with almost-zero skepticism (almost-zero error) after repetitious corrections (updating the synaptic weights).* This hypothesis has been made by the author's imagination. This process is called *learning.* Error feedback may simulate the correction process of the brain by using the discrepancy between the actual signal (sensed one from outside for human) and the neural output (memory in human) to learn about a system (an external environment for human). The effect of error feedback in ANNs is shown in Figure 31 in terms of the identification precision and speed.

New feedforward synaptic weights are also introduced to the RNNs for the interconnections between layers, called the inter-layer as shown in Figure 14. This additional link creates meaningful connections from the hidden layers to the output layer. These inter-layer connections directly deflect the influence of inputs in the hidden layers to the output layer. Generally external inputs and internal (recurrent) inputs in the hidden layers pass to the next layer through internal synaptic weights in each layer. If each layer has (unity) time-delay units, the output layer always gets the old data from previous layers depending on how many layers a signal passes through. Therefore, it loses the influence of the most recent inputs data automatically in the conventional multilayer feedforward neural nets resulting in narrow time-order for time difference equation. Therefore, inter-layer synaptic weights play a role to supply the most recent external-input data to the output layer directly leading to wider time-order in terms of time difference equation. In conventional linear models of ARX and ARMAX, the increment of time-order causes the number of parameters to grow proportionally, which results in a drastically increased load to the estimation algorithm. However, the computation load, which is produced by the addition of inter-layer links for the purpose of getting more fresh data from the hidden layer, is shared with layers in the SERNN. Since the training process in SERNN is carried out layer by layer, the use of inter-layer synaptic weights can result in the reduced computation load as well as similar time-order effects. In the SERNN model, time-delay unit need not always confine to *unity* delay in each layer. The cascaded unity-delay in each layer can give the model more flexible (wider) time-order for both external input and recurrent output signals.

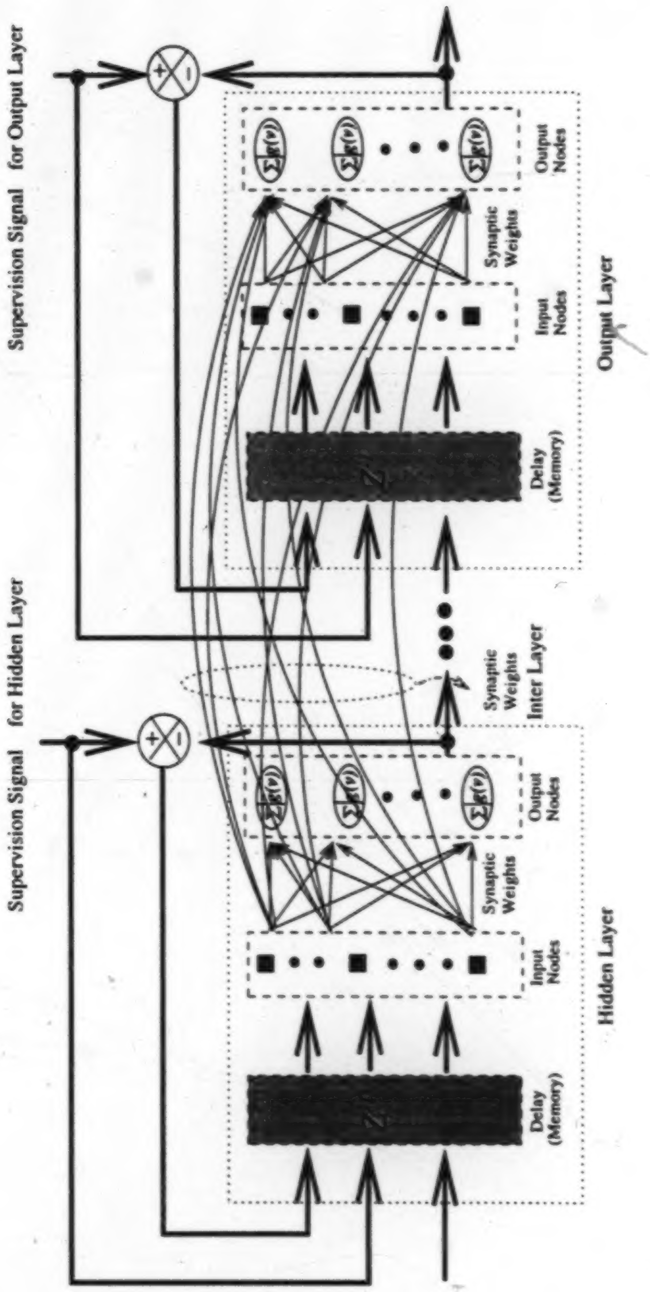

Supervision Signal for Output Layer
Supervision Signal for Hidden Layer
+
-
Output Nodes
Synaptic Weights
Input Nodes
Delay (Memory)
Output Layer
Synaptic Weights Inter Layer
Hidden Layer

Figure 14: Proposed Supervision & Error RNN (SERNN)

4.6 Architectural Techniques

The following sections introduce fundamental concepts and basic ways developed in this thesis for the new recurrent architecture. This architecture is developed by using mostly empirical knowledge and heuristics. This neural architecture has an objective of fast, robust and accurate on-line system identification for the various dynamic and unknown systems. In this development, the following architectural nomenclature is used in conjunction with Figures 15 ~ 26.

- Layer: There are three layers; ***hidden layer***, ***output layer*** and ***virtual layer***. (Input layer is called one of the hidden layer in the following architecture since the input layer is working for simple data-distributing points in the left-most hidden layer.)

  1. *Output Layer (OL)*: is the right-most layer in a multilayer neural network, which consists of input nodes, (internal) forward synaptic weights and output nodes. (See Figure 15.)
  2. *Hidden Layer (HL)*: refers to all layers on the left-side of the output layer, where forward direction is represented from left to right. HL consists of input nodes, (internal) forward synaptic weights and output nodes. (See Figure 15.)
  3. *Virtual Layer (VL)*: refers to all layers outside of a concerned neural network. A VL consists of only one column of nodes, which can actually be regarded as new external data points which play a role to supply the available information from outside to the network. (See Figures 20, 21, 22, 23, 24, 25.) In this research, two kinds of VL are developed:
     - (a) *VL* of supervision signals: These data points are formed by feedback of teaching (supervision) signals in each layer. This new input data-source creates recurrent synaptic weights. (See Figures 21, 22.)
     - (b) *VL* of residual errors: These data points are formed from residual errors between teaching signals and outputs in each layer. This new

data source can replace the threshold or bias, and also create recurrent synaptic weights. (See Figures 23, 24, 25.)

- *Input Node (IN)*: This is a data-point receiving signal in hidden and output layers. (See Figure 15.)
- *Output Node (ON)*: This is a point of processing-unit sending signal out in hidden and output layer. (See Figure 15.)
- *Virtual Node (VN)*: This is a simple data-point (distributor) in virtual layer not having IN and ON. (See Figures 20, 21, 22, 23, 24, 25, where VNs are different data points depending on Figures.)
- *Forward (FW)*: This indicates the direction of signal flow passing through from the external input to the output in the output layer. FW links consist of *internal* forward synaptic weights in each layer as shown in Figure 15 and *inter-layer* forward synaptic weights between layers as shown in Figures 16, 17.
- *Backward (BW)*: Also called *recurrent* or *feedback*. BW links consist of *internal* backward synaptic weights in hidden layers and output layer as shown in Figure 18, and *inter-layer* backward synaptic weights between layers as shown in Figures 19, 20, 21, 22, 23, 24, 25.
- *Synaptic Weight (SW)*: This is a connection from IN to ON or vice versa in same layer and between layers, which is an adjustable variable.
- *Input*: This is a signal flowing into the input nodes in the hidden and the output layers.
- *Output*: This is a signal going out from the output nodes in the hidden and the output layers.
- *Time-Delay Unit*: This means *memory* in discrete-time processing, which makes a signal have the time-history such as oldest, older, old, etc., and present signals with respect to sampling instance. Each layer has unity time-delay in this research as shown in Figure 26, where time-order is changed whenever a signal passes through each layer until it gets to the output nodes in the output layer.

- *OL-ON*: Output Nodes (ON) in Output Layer (OL).
- *OL-IN*: Input Nodes (IN) in Output Layer (OL).
- *HL-ON*: Output Nodes (ON) in Hidden Layer (HL).
- *HL-IN*: Input Nodes (IN) in Hidden Layer (HL).
- *VL-VN*: Virtual Nodes (VN) in Virtual Layer (VL).
- *OL-FW-SW*: ForWard (FW) Synaptic Weights (SW) in Output Layer (OL).
- *OL-BW-SW*: BackWard (BW) Synaptic Weights (SW) in Output Layer (OL).
- *HL-FW-SW*: ForWard (FW) Synaptic Weights (SW) in Hidden Layer (HL).
- *HL-BW-SW*: BackWard (BW) Synaptic Weights (SW) in Hidden Layer (HL).
- *IL-FW-SW*: ForWard (FW) Synaptic Weights (SW) in Inter Layer (IL).
- *IL-BW-SW*: BackWard (BW) Synaptic Weights (SW) in Inter Layer (IL).

Feedforward Synaptic Weights

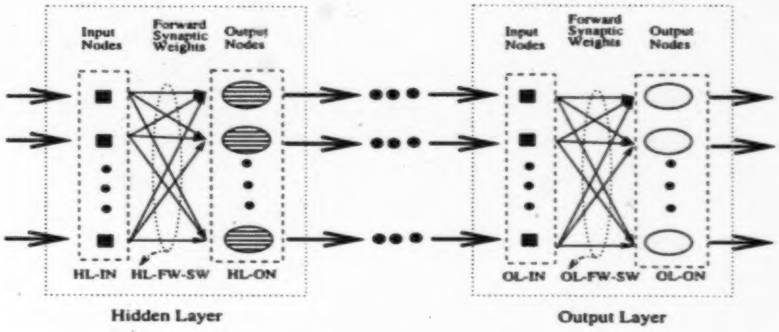

Input Nodes
Forward Synaptic Weights
Output Nodes
HL-IN
HL-FW-SW
HL-ON
Hidden Layer
Input Nodes
Forward Synaptic Weights
Output Nodes
OL-IN
OL-FW-SW
OL-ON
Output Layer

Figure 15: From OL-IN To OL-ON and From HL-IN To HL-ON

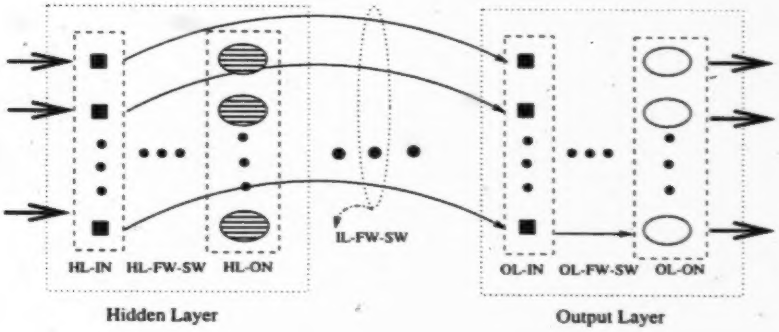

HL-IN
HL-FW-SW
HL-ON
Hidden Layer
IL-FW-SW
OL-IN
OL-FW-SW
OL-ON
Output Layer

Figure 16: From HL-IN To OL-IN For Inter-Layer

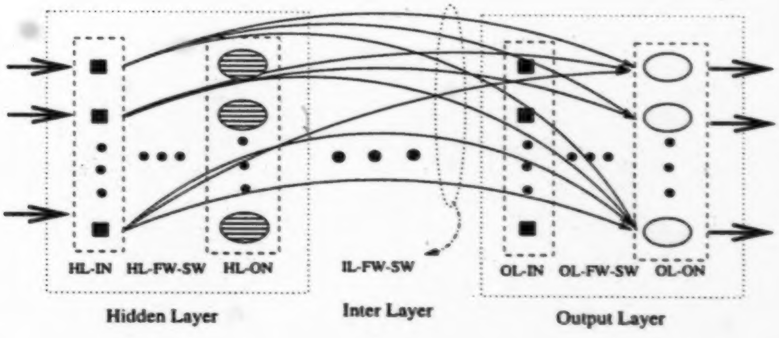

HL-IN
HL-FW-SW
HL-ON
Hidden Layer
IL-FW-SW
Inter Layer
OL-IN
OL-FW-SW
OL-ON
Output Layer

Figure 17: From HL-IN To OL-ON For Inter-Layer

Backward (Recurrent) Synaptic Weights

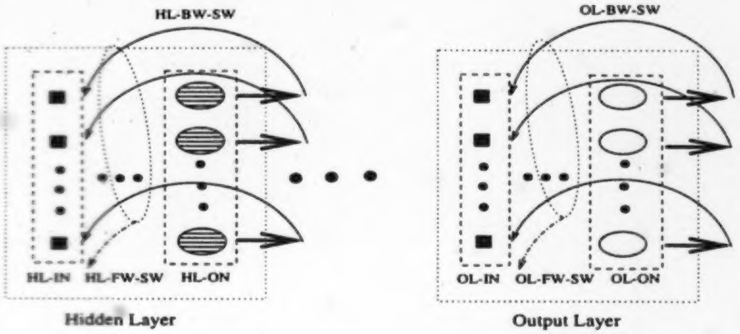

HL-BW-SW
OL-BW-SW
HL-IN
HL-FW-SW
HL-ON
OL-IN
OL-FW-SW
OL-ON
Hidden Layer
Output Layer

Figure 18: From OL-ON To OL-IN and From HL-ON To HL-IN

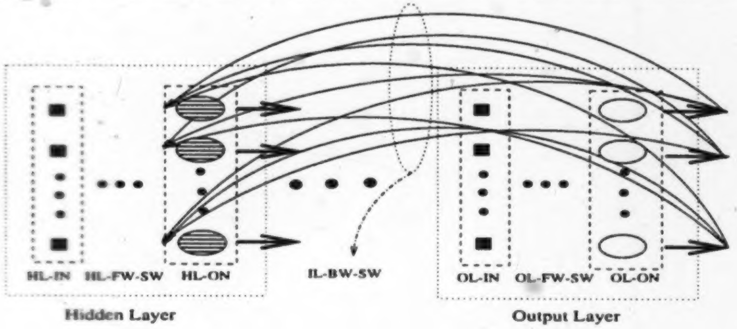

HL-IN
HL-FW-SW
HL-ON
IL-BW-SW
OL-IN
OL-FW-SW
OL-ON
Hidden Layer
Output Layer

Figure 19: From OL-ON To HL-ON For Inter-Layer

Backward Synaptic Weights Using Virtual Layer

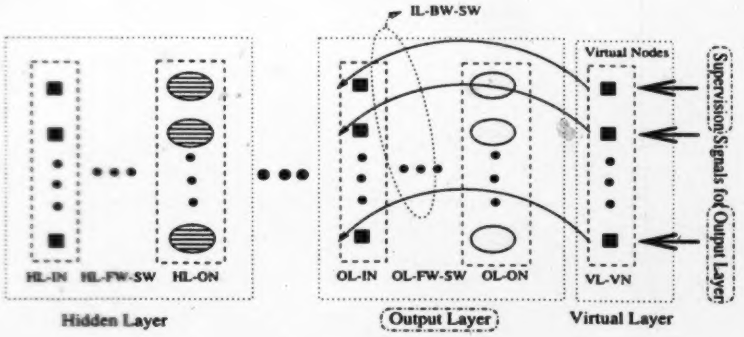

IL-BW-SW
Virtual Nodes
Supervision Signals for Output Layer
HL-IN
HL-FW-SW
HL-ON
OL-IN
OL-FW-SW
OL-ON
VL-VN
Hidden Layer
Output Layer
Virtual Layer

Figure 20: From VL-VN(Supervision) To OL-IN

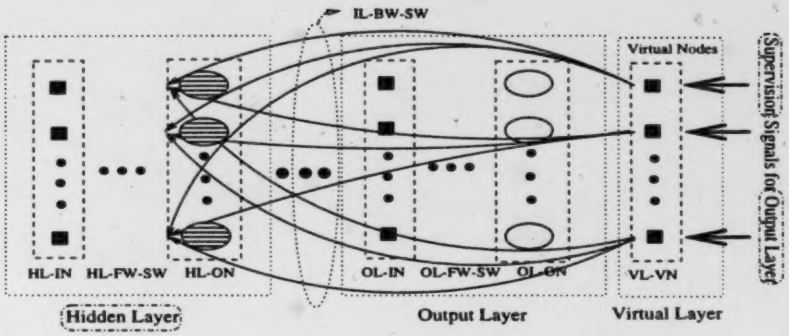

IL-BW-SW
Virtual Nodes
Supervision Signals for Output Layer
HL-IN
HL-FW-SW
HL-ON
OL-IN
OL-FW-SW
OL-ON
VL-VN
Hidden Layer
Output Layer
Virtual Layer

Figure 21: From VL-VN(Supervision) To HL-ON

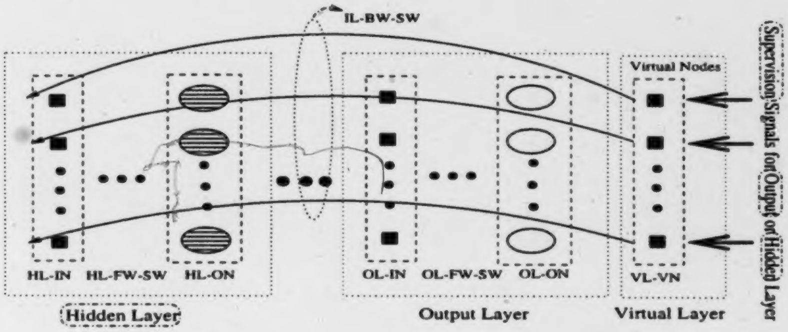

IL-BW-SW
Virtual Nodes
Supervision Signals for Output or Hidden Layer
HL-IN
HL-FW-SW
HL-ON
OL-IN
OL-FW-SW
OL-ON
VL-VN
Hidden Layer
Output Layer
Virtual Layer

Figure 22: From VL-VN(Supervision) To HL-IN

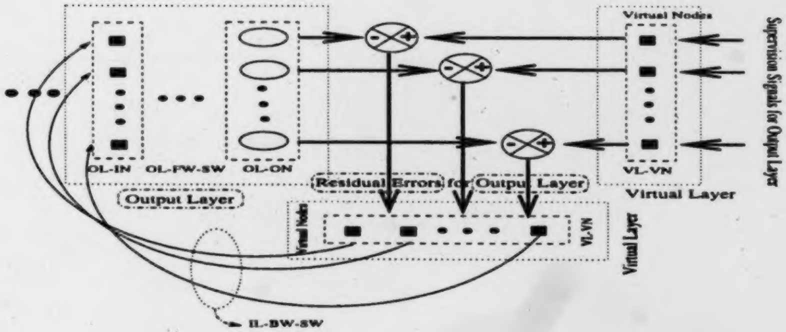
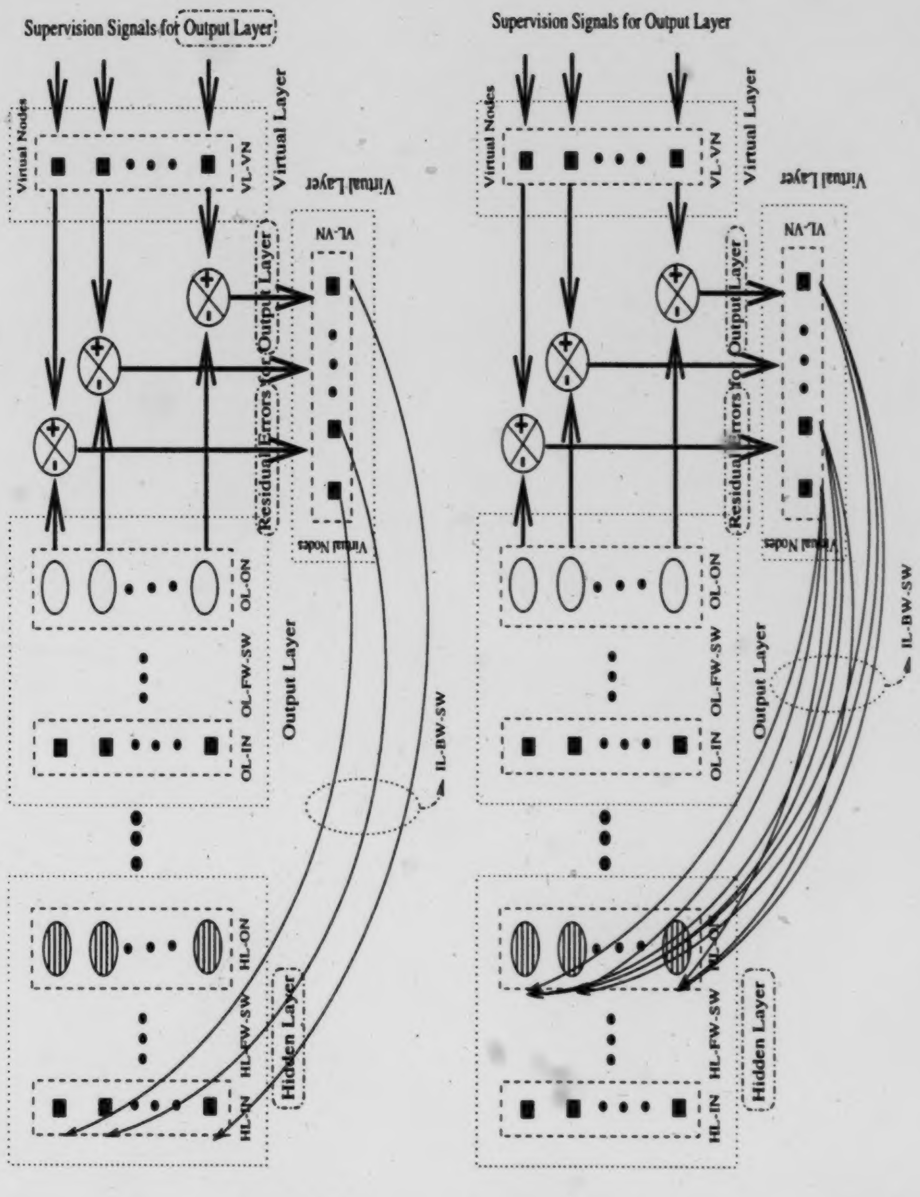

Virtual Nodes
Supervision Signals for Output Layer
-
+
OL-IN
OL-FW-SW
OL-ON
VL-VN
Virtual Layer
Output Layer
Residual Errors for Output Layer
Virtual Nodes
VL-VN
Virtual Layer
IL-BW-SW

Figure 23: From VL-VN(Output-Layer Error) To OL-IN


Supervision Signals for Output Layer
Virtual Nodes
VL-VN
Virtual Layer
Residual Errors for Output Layer
OL-ON
OL-FW-SW
OL-IN
Output Layer
HL-ON
HL-FW-SW
HL-IN
Hidden Layer
IL-BW-SW
Virtual Nodes
VL-VN
Virtual Layer
Supervision Signals for Output Layer
Virtual Nodes
VL-VN
Virtual Layer
Residual Errors for Output Layer
OL-ON
OL-FW-SW
OL-IN
Output Layer
HL-ON
HL-FW-SW
HL-IN
Hidden Layer
IL-BW-SW
Virtual Nodes
VL-VN
Virtual Layer

(a) From VL-VN (Output-Layer Error) To HL-IN

(b) From VL-VN (Output-Layer Error) To HL-ON

Figure 24: From VL-VN to (a) HL-IN and (b) HL-ON

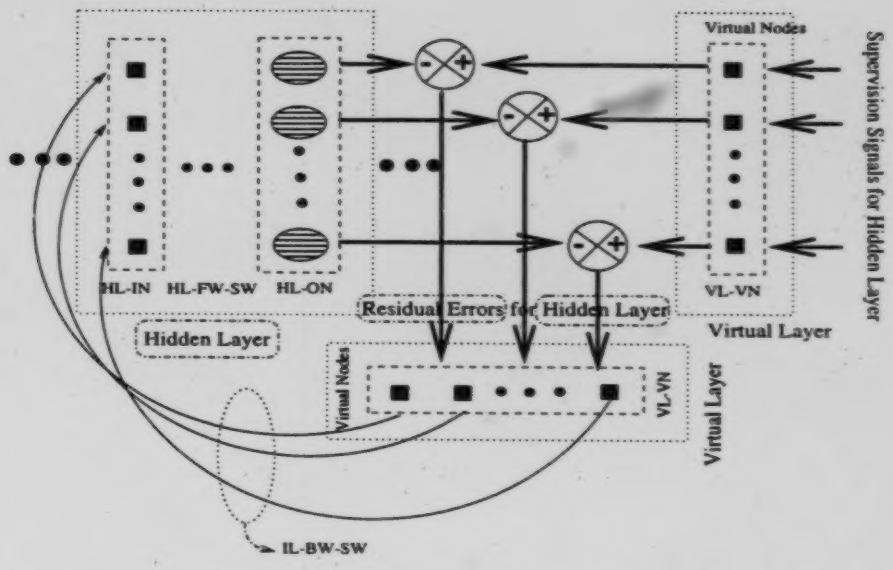

Virtual Nodes
Supervision Signals for Hidden Layer
-
+
HL-IN
HL-FW-SW
HL-ON
VL-VN
Residual Errors for Hidden Layer
Hidden Layer
Virtual Layer
Virtual Nodes
VL-VN
Virtual Layer
IL-BW-SW

Figure 25: From VL-VN(Hidden-Layer Error) To HL-IN

Time-Delay Units for Memory

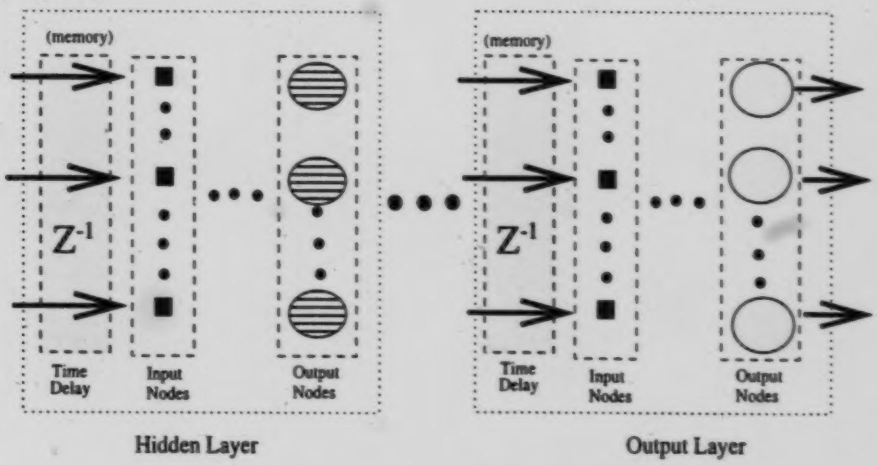

(memory)
Z^-1
Time Delay
Input Nodes
Output Nodes
Hidden Layer
(memory)
Z^-1
Time Delay
Input Nodes
Output Nodes
Output Layer

Figure 26: Unity Time-Delay For Memory

4.6.1 A New Neuron Model Of Error-Feedback

In the neural device (neuron) design, the *bias* is used such that it keeps a constant level of activation in the absence of input and must be chosen to avoid a dead zone [1]. Also, the introduction of the bias to the neural network design can contribute significantly to the mean-squared error between the desired response and the neural response [7], where the bias is said at least to be *harmless*. In essence, a bias is needed for designing each specific application. A practical way of achieving such an objective is to use a constrained network architecture, which usually performs better than a general-purpose architecture. For example, the constraints and therefore the bias may take the form of prior knowledge built into the network design using: (a) *weight sharing* where several synapses of the network are controlled by a single weight; (b) *local receptive fields* assigned to individual neurons in the network, as demonstrated in the application of a multilayer perceptron to optical character recognition problem [19].

However, the clear effect of bias and a global methodology to determine its value do not seem to exist to date. In a general neuron model, the *threshold* (-1.0) or *bias* $(+1.0)$ applied externally has the effect of lowering and increasing the net input of the activation function respectively [8] as shown in Figure 27.

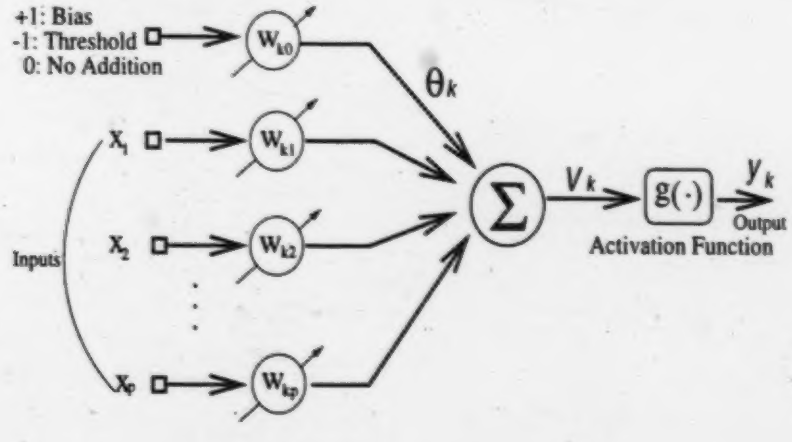


Figure 27: A Conventional Neuron Model

However, it is difficult for designers to determine their magnitude and sign which depend on the system under consideration and to find their physical interpretation. In this thesis, a new neuron model has an architecture such that:

- the threshold or the bias takes its value from error feedback with a time-delay unit between actual (desired or supervisory) signal and model output for the supervised training method.
- synaptic weights are added for the above recurrent error signal. This is shown in Figure 28.

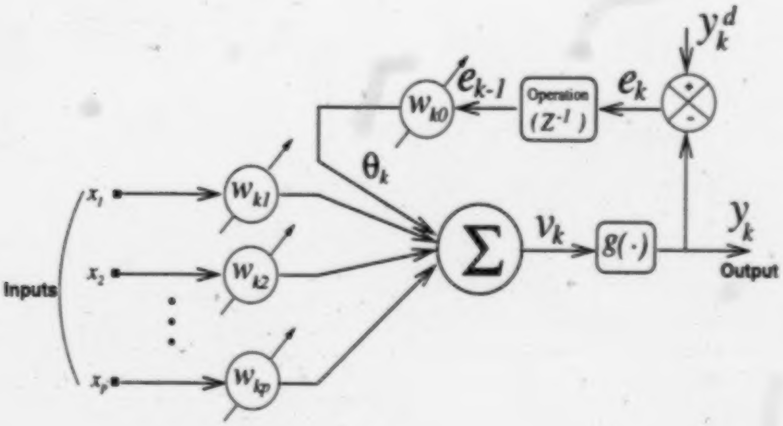


Figure 28: A New Neuron Model

The new neuron model having the feedback of error signal can make an automatic decision on the sign and magnitude for threshold and bias values, and can avoid the manual selection of -1 or $+1$ by a designer such as:

$$\boldsymbol{\theta}_k = \mathbf{W}_{k0} \cdot \mathbf{e}_{k-1}$$

The architecture based on the new neuron model has been shown in Figures 22, 24, 25. The error feedback is expected to make a direct contribution to fast training performance because it creates a new and meaningful input source to the neural architecture, which leads to fast identification capability [1]. Its expected effect is illustrated in Figures 29 which is exactly the same as shown in Figure 28.

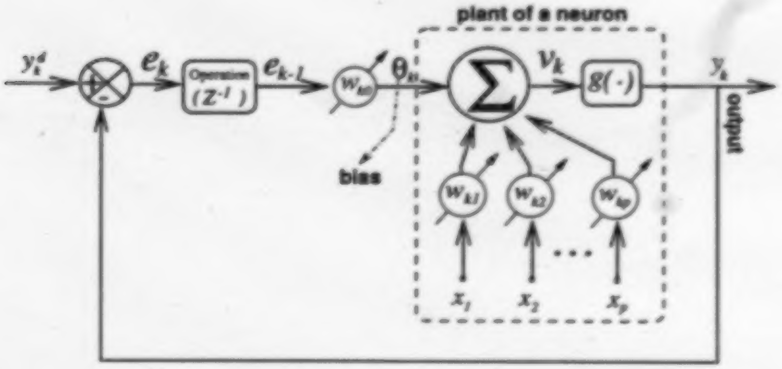


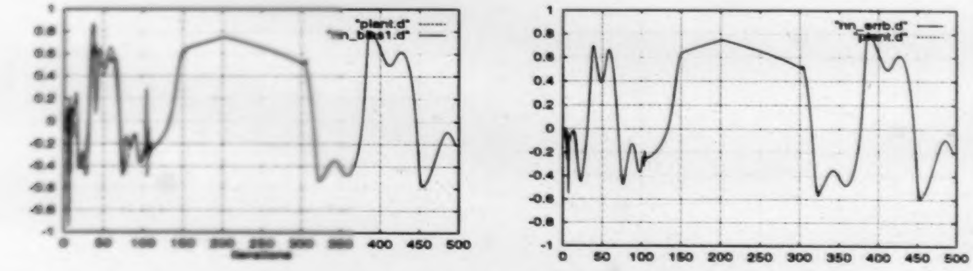
Figure 29: A New Neuron Model Observed By Feedback Control

In Figure 29, θ_k replaces the bias signal in a conventional neuron model with $\theta_k = w_{k0} \cdot e_{k-1}$, but it plays a role of (adaptive) control signal for a certain neuron plant in the new neuron model when the general training algorithm updates the bias synaptic weight w_{k0} (including $w_{k1}, w_{k2}, \dots, w_{kp}$) to minimize the error of e_k. The teaching signal y_k^d and the input data $x_1, x_2, \dots, x_p$ are given as a training set for the neural networks. For the purpose of comparison between the conventional and the new neuron models, a nonlinear plant is simulated and identified as shown in Figures 30 and 31. The **left-hand side figures** show the identification results based on the **conventional neuron model**, while the **right-hand side figures** clearly show the **faster identification performance [2] by the new neuron model**. Both neural networks built by the conventional and new neuron models use the same on-line training method of the standard RLS algorithm with a clear supervision signal in the hidden layer, whose development is explained in the next chapter. The new neuron model solves the problem of the bias/threshold signal selection effectively and turns out to be very useful for fast, accurate and robust neural system identification through the following simulations in this research.

---

[1] Published in IEEE International Conference on Systems, Man and Cybernetics, Vol.1, Oct.14-17, pp.356-50,1996.

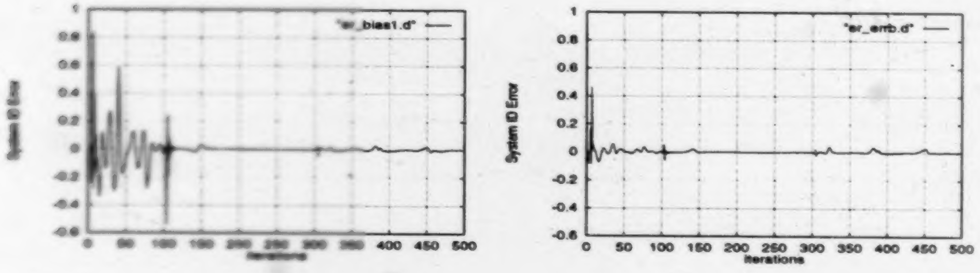
[2] Published in Journal of KIEE *(Korean Institute of Electrical Engineers)*, July, 1997



(a) based on a conventional neuron

(b) based on a new neuron

Figure 30: Real-Time System Identification (Actual Output + Model Output)



(a) based on a conventional neuron

(b) based on a new neuron

Figure 31: Identification Error

4.6.2 A New Feedback Of Supervision Signal

In all neural architectures published in the literature, the self-feedback of the output from the hidden and/or output-layers (internal feedback) has been done for the recurrent neural structure. In this research, instead, the feedback of the supervision signal from outside the neural model, that is output from the plant is heuristically proposed. Both the internal (self) and external feedback creates new inputs to the networks and thereby computes new synaptic weights for them. This novel recurrent neural architecture is made on the assumption that the external feedback is able to transfer more accurate information about the plant to network inputs than the internal feedback can do alone. It may be mentioned that the memory capability of networks comes from the recurrent connections described in the Section 4.2. Especially, at the beginning of the on-line identification, due to poor initial-weight values, the network output has a greater identification error, i.e., plant-output tracking error. A question arises whether it is prudent that the poor neural outputs at the beginning of training should be fed as inputs in order to reduce the residual (modeling) error for the next learning step. This self-feedback of poor neural model outputs may lead to a vicious cycle of mal-information. Based on the above heuristics, the feedback of supervision (desired) signals is designed for the recurrent neural net architecture for the purpose of developing a fast, accurate, and robust system identification. This architecture of the supervision feedback has been shown in the Figure 19, 20, 21.

4.6.3 Inter-Layer Forward Connections

As a further development in this thesis, the inter-layer forward synaptic weights are inserted additionally between the input-nodes in the hidden-layer (HL-IN) and the output-nodes in the output-layers (OL-ON) as shown in Figure 14. The inter-layer connections can create wider time-order in the time difference equation for the developed multi-layer recurrent neural network called the SERNN shown in Section 4.7. Each output and hidden layer is designed to have unity time-delay units at the input nodes. Unity time-delay occurs whenever signals pass through a layer. In the absence of inter-layer synaptic weights, only two-unity time-delayed external input signals in the hidden-layer reach the output-nodes in the output-layer in case of two-layer neural nets (See Figure 26). Therefore, unity time-delayed external inputs in the hidden layer never go to the output-layer. By adding the ***inter-layer forward synaptic connections***, however, the output-layer is able to get both unity and dual time-delayed signals simultaneously, which leads to a larger time-order. With the multi-layer architecture consisting of more layers and inter-layers, the time-order can be theoretically increased to infinity as in a conventional ARX and ARMAX (linear parametric) models. The inter-layer connection has a clear advantage over the non inter-layer neural architecture with respect to the number of synaptic weights being trained at a time since ANNs' weights training is performed by means of layer by layer, while the parameters in the conventional linear model are all estimated at one time. It is well known that as the number of parameters to be estimated increases, the estimation performance deteriorates [11]. In the time-delayed multi-layer recurrent neural architecture, it is important to recognize that the time-order in the difference equation should not be sacrificed under the name of the multi-layer networks. The ways of inter-layer forward connection have been shown in Figure 16 and 17.

4.7 Schematic Diagram of SERNN

The following two schematic diagrams show typical network structures built by the architecture proposed in the Section 4.6. The single-layer architecture for the SERNN is shown in Figure 32 which is built as shown in Figures 15, 20, 23, 26. The multi-layer architecture for the SERNN is also shown in Figure 33 which is built using components as shown in Figures 15, 17, 20, 22, 23, 25, 26. Different combinations of the proposed architectural techniques can create other functional nets depending on the application. This section introduces the single-layer (3-input 3-output) and multi-layer (multi-input multi-output) SERNN.

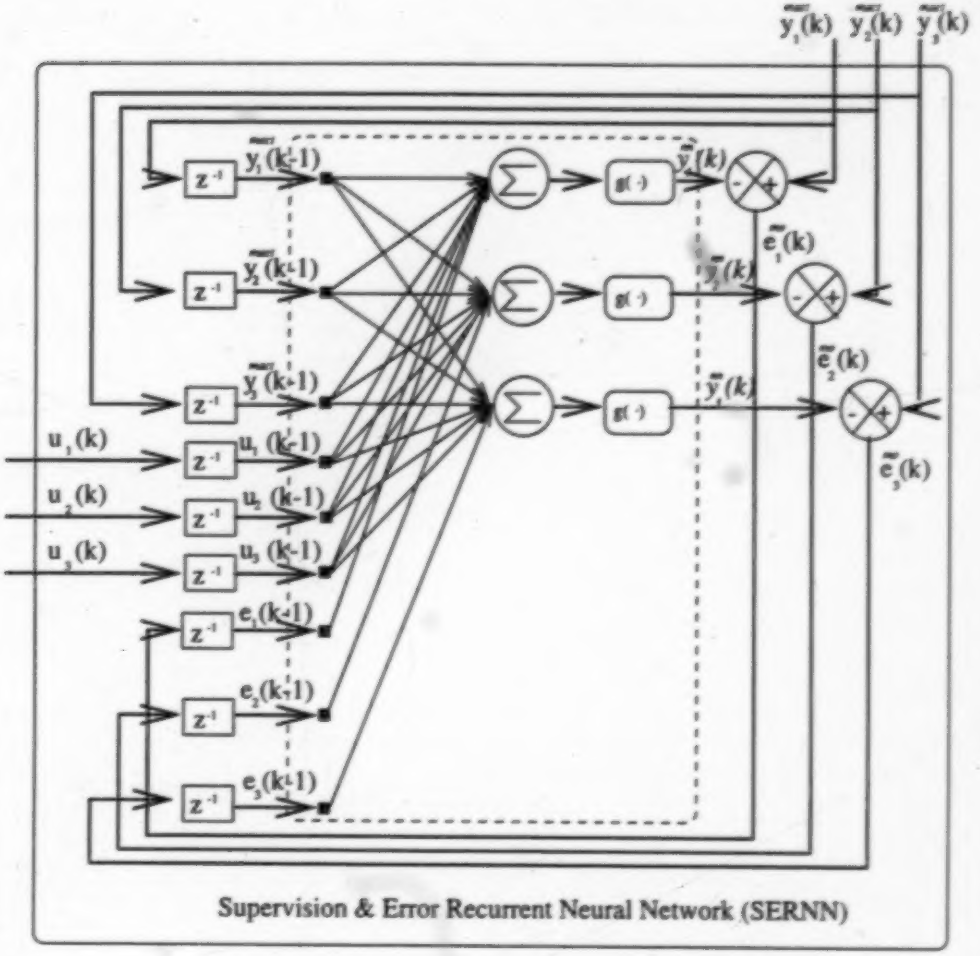

y_1(k) y_2(k) y_3(k)
z^-1
y_1(k-1)
y_2(k-1)
y_3(k-1)
u_1(k)
u_2(k)
u_3(k)
u_1(k-1)
u_2(k-1)
u_3(k-1)
e_1(k-1)
e_2(k-1)
e_3(k-1)
Σ
g(·)
y_1(k)
y_2(k)
y_3(k)
e_1(k)
e_2(k)
e_3(k)
-
+
Supervision & Error Recurrent Neural Network (SERNN)

Figure 32: Single-Layer Schematic of SERNN

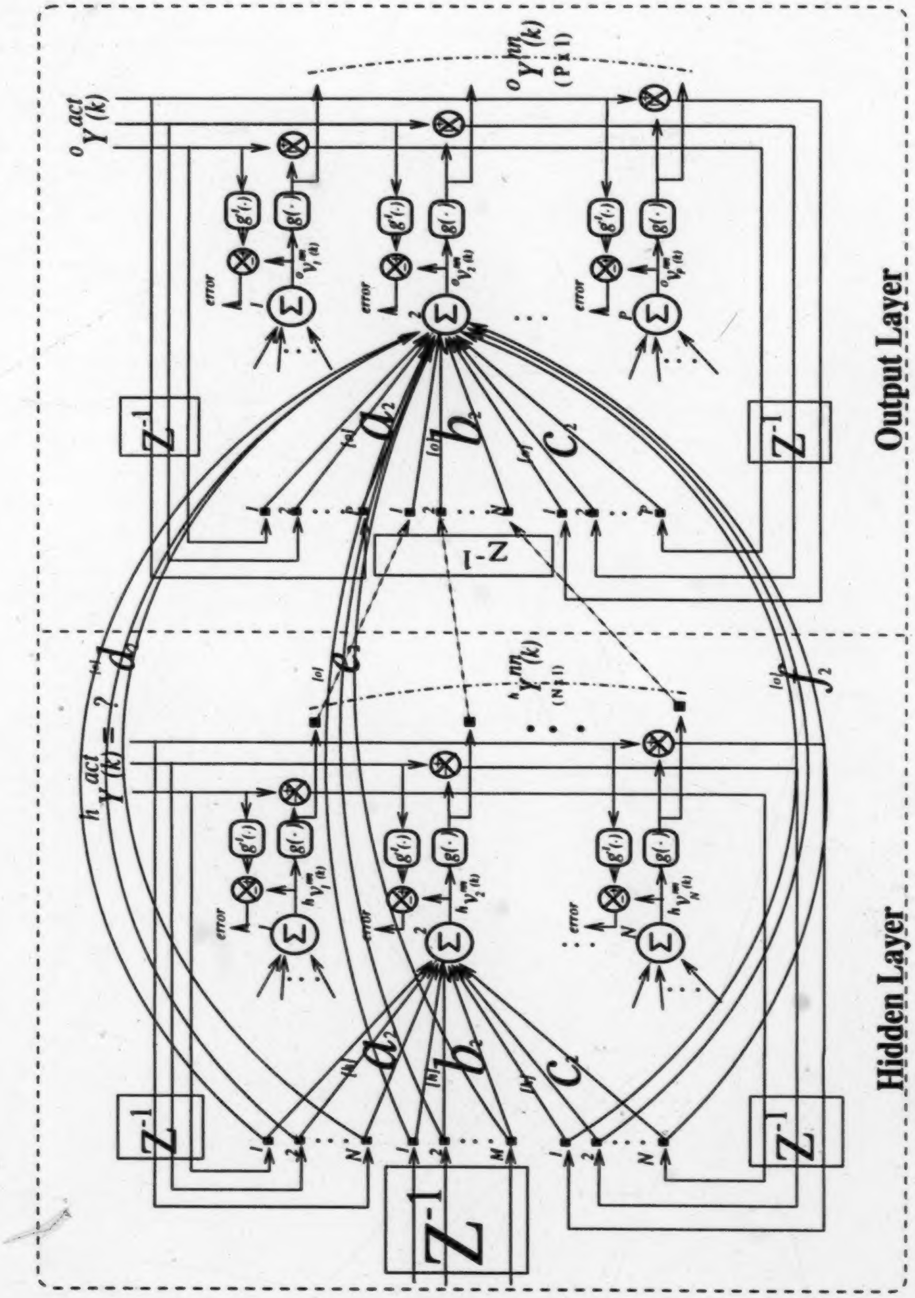

Output Layer
Hidden Layer
Z-1
Z-1
Z-1
Z-1
Z-1
Z-1
error
g(·)
g'(·)
Σ
o act Y(k)
o nn Y(k) (P x 1)
h act Y(k) = ?
h nn Y(k) (N x 1)

Figure 33: Multi-Layer (Detailed) Schematic of SERNN

4.8 Remarks On the Developed SERNN

The architectural features of the proposed SERNN are as follows:

- One layer is divided into the three parts of the *input nodes*, the *internal forward synaptic weights* and the *output nodes*. This allows a designer to enhance the chance of observation for more connections and to make systematic links instead of **spaghetti**[3] **synaptic weights**. The creation of the virtual layer can be regarded as a result of the systematic link methodology. The proposed architectural techniques are useful to design diverse neural topologies for different applications.

- The bias of a simple constant in a conventional neuron model is replaced with **error feedback** in a new neuron model and a synaptic weight is added to this. The bias of error-feedback is actually a time-varying signal, and thereby a **time-varying bias**. The new bias can play a role of a (adaptive) controller inside each neuron by the added synaptic weight with the time-varying bias, because the weight for bias is updated continuously to minimize the error between the desired response and the neural response. This method of automatic bias can make a significant contribution to fast tracking system identification as shown in Figure 31.

- The **supervision-feedback** is developed in designing the SERNN architecture, while the conventional RNNs use the self-feedback of neural output. It is presumed that the self-feedback architecture for the RNNs is less desirable than the supervision-feedback. This assumption is based on the fact that the neural mapping performance gets poor at the beginning of neural training process unless the optimal initial weights are selected. The feedback of a poorly mapped neural output may be a *vicious-cycle of the poor output* which may lead to the instability at the beginning of training process. However, the proposed supervision-feedback can take more accurate information of the plant under consideration by the recurrent actual plant output, because the aim is to minimize the error between the actual plant output and the neural model output, in other words, to track the actual output by the model output.

---

[3] Means unstructured or tangled synaptic links. In the Software Engineering, the spaghetti program results from frequent use of the *go to* command and should be avoided.

- The SERNN architecture always requires clear supervision signals to train each layer effectively. The BP algorithm is not adequate to train the novel SERNN network because BP does not have clear teaching signals for the hidden layer, but only ones for the output layer. To effect teaching the hidden layers, BP uses the backpropagating of error occurring in the output layer to the hidden layers. This may cause the slow learning because all hidden layers have the same supervisors (i.e., the same error-gradient with different learning rates) as that in the output layer. A question arises whether it is reasonable with regard to the multi-layer ANNs topology which is generally cascaded. On this matter, the novel teaching signals for the hidden layers are derived by the L_2-norm optimization method in the next chapter where the novel on-line training algorithm called MRLS algorithm is also developed. The combination of the SERNN architecture in this chapter and the MRLS training algorithm with clear teaching signals presented in the next chapter leads to the development of a fast, robust and accurate neural system identifier.

Bibliography

[1] James A. Anderson. *An Introduction to Neural Networks.* The MIT Press, Cambridge, Massachusetts, London, England, 1995.

[2] V. Braitenberg. Is the cerebellar cortex a biological clock in the millisecond range? In C.A. Fox and eds. R.S. Snider, editors, *In The Cerebellum. Progress in Brain Research 25*, pages 334–346. Amsterdam:Elsevier, 1967.

[3] V. Braitenberg. *On the Texture of Brains.* Springer-Verlag, New York, 1977.

[4] V. Braitenberg. *Two views of the cerebral cortex.* Springer-Verlag, New York, 1986. In Brain Theory (G. Palm and A. Aertsen, eds.).

[5] J. Elman. Representation and structure in connectionist models. *Machine Learning*, vol.7(no.2–3), 1989.

[6] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, vol.14:pp.179–211, 1990.

[7] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and bias/variance dilemmma. *Neural Computation*, vol.4:pp.1–58, 1992.

[8] Simon Haykin. *Neural Networks : A comprehensive Foundation.* Macmillan College Publishing Company Inc., New York, 1994.

[9] J.J. Hopfield and D.W. Tank. Neural computation of decision in optimization problems. *Biol. Cybern.*, vol.52:pp.141–152, 1985.

[10] J.J. Hopfield and D.W. Tank. Simple neural optimization networks: an a/d converter, signal decision circuit, and a linear programming circuit. *IEEE Trans. Circ. Syst.*, CAS-33:pp.533–41, 1986.

[11] K.J. Hunt. Stochastic optimal control theory with application in self-tuning control. In Edited by M. Thomas and A. Wyner, editors, *Lecture Notes in Control and Information Sciences (Series: 117)*, Berlin; New York, 1989. Springer-Verlag.

[12] Jean-François Jodouin. Putting the simple recurrent network to the test. *1993 IEEE Int. Conf. on Neural Networks*, vol.12:pp.1141–1146, 1993.

[13] T.A. Johansen and B.A. Foss. A narmax model representation for adaptive control based on local models. *Modeling, Identification and Control*, vol.13(no.1), 1992.

[14] M.I. Jordan. Serial order: A parallel distributed processing approach. Technical Report No.8604, University of California, Institute for Cognitive Science, San Diego, 1986.

[15] C.-C. Ku and K.Y. Lee. System identification and control using diagonal recurrent neural networks. *Proc. Amer. Control Conf.*, vol.1:pp.545–549, 1992.

[16] C.-M. Kuan, K. Hornik, and H. White. A convergence result for learning in recurrent neural networks. *Neural Computation*, vol.6:pp.420–440, 1994.

[17] Q.M. Lam, R. Chipman, and J. Sunkel. Mass property identification: A comparison study between extended kalman filter and nero-filter approaches. *Proc. AIAA Guidance Navigation Control*, 1991.

[18] K.J. Lang and G.E. Hinton. The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-Cs-88-152, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[19] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. handwritten digit recognition with a back-propagation network. In ed. D.S. Touretsky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404, San Mateo, CA., 1990. Morgan Kaufmann.

[20] Seong-Whan Lee and Hee-Heon Song. A new recurrent neural-network architecture for visual pattern recognition. *IEEE Trans. on Neural Networks*, vol.8(no.2):pp.331–339, March 1997.

[21] Lennart Ljung. *System Identification.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1987.

[22] Lennart Ljung and Torsten Soderstrom. *Theory and Practice of Recursive Identification.* The MIT Press, Cambridge, Massachusetts, London, England, 1983.

[23] R. Miller. Representation of brief temporal patterns, hebbian synapses, and the left-hemisphere dominance for phoneme recognition. *Psychobiology*, vol.15:pp.241–247, 1987.

[24] D. Neumerkel, R. Murray-Smith, and H. Gollee. Modelling dynamic processes with clustered time-delay neurons. *Proceedings of 1993 International Joint Conference on Neural Networks, Nagoya, Japan*, vol.2:pp.1765–1768, 1993.

[25] F.J. Pineda. Generalization of back propagation to recurrent and higher order neural networks. In D.Z. Anderson (Ed.), editor, *Neural information processing systems*, New York, 1988. American Institute of Physics.

[26] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group. *Parallel distributed processing (PDP): Exploration in the microstructure of cognition*, volume vol.1. MIT Press, Cambridge, MA, 1986.

[27] R. Shoureshi, R. Chu, and M. Tenorio. Neural networks for system identification. *Proc. Amer. Control Conf.*, vol.1:pp.916–921, 1989.

[28] M.F. StJohn and J.L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, vol.46:pp.217–257, 1990.

[29] A. Stolke. Learning feature-based sematics with simple recurrent networks. Technical Report TR-90-015, International Computer Science Intitute, 1990.

[30] W.S. Stornetta, T. Hogg, and B.A. Huberman. A dynamical approach to temporal pattern processing. *Proceedings of the IEEE Conference on Neural Information Processing Systems*, 1987. Denver, CO.

[31] D.W. Tank and J.J. Hopfield. Neural computation by concentrating information in time. *Proceedings of the IEEE International Conference on Neural Networks*, 1987. San Diego, CA.

[32] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-37:pp.328–339, 1989.

[33] R.L. Watrous and L. Shastri. Learning phonetic features using connectionist networks: An experiment in speech recognition. *Proceedings of the IEEE International Conference on Neural Networks*, 1987. San Diego, CA.

[34] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. Technical Report No.8805, Institute for Cognitive Science, University of California, San Diego, 1988.

[35] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, vol.1(no.2):pp.270–280, 1989.

Chapter 5

Real-Time Multi-layer Neural Networks Training Algorithm

5.1 Introduction

Real-time (on-line) system identification refers to the process of developing a neural network model described in Chapter 4. It involves estimating time-varying synaptic weights recursively at each sampling period kT, by observing input and output measurements for the system being considered. In this chapter, the typical training algorithms for neural networks are discussed, and a fast, robust and accurate on-line training algorithm is developed to improve the performance of dynamic system identification based on recurrent neural nets.

In the literature a variety of training algorithms have been proposed, such as error backpropagation (BP), competitive learning, Boltzmann learning, etc.. Each learning scheme is useful for a specific type of application. The main problem in training the multi-layer ANNs has been the ***absence of a desired output*** (a teaching signal) for the hidden layers. This can be solved by the error BP algorithm in the form of ***gradient descent*** optimization. (Also called ***generalized delta rule***, and similar to ***steepest descent***.) In BP, the error for the hidden layer is calculated by the propagation of errors back from the output layer to the hidden layer, and from this hidden layer to the previous hidden layer, and so forth. It can be shown that the synaptic weight adjustment by the BP algorithm minimizes the summed square of the errors between the desired and the actual outputs over the entire training set. The mathematical details of the algorithm can be found in [16]. The error BP algorithm is a systematic procedure to train the multi-layer feedforward/recurrent ANNs.

Here, the gradient-based training algorithms are briefly reviewed, and these formulate the gradients of the cost functional with respect to the various parameters of the neural architecture, i.e., synaptic weights, for the minimization process. In a variety of the literature, major efforts have been devoted to two aspects: the efficacy of their computation (in the past) and efficient calculation of the gradients of a system's output with respect to different parameters of the neural architecture. The following shows their efforts for the latter cases of above two aspects in the BP algorithm clones.

[26] presented a scheme in which the gradients of an error functional with respect to network parameters were calculated by direct differentiation of the neural activation dynamics. This approach is computationally very expensive and scales poorly to large systems. The inherent advantage of the scheme is the small storage capacity required. [14] described a variational method which yields a set of linear ordinary differential equations for backpropagating the error through the system. These equations, however, need to be solved backward in time, and require storage of variables from the network activation dynamics, thereby resulting in less desirable effects. [22] has suggested a framework which, in contradiction to [14]'s formalism, enables the error propagation system of equations to be solved forward in time, concomitantly with the neural activation dynamics. A drawback of this preliminary approach comes from the fact that these equations have to be analyzed in terms of distributions, which preclude straightforward numerical implementation. [18] proposed a conceptual algorithm based on Lagrange multipliers. However, this algorithm has not yet been validated by numerical simulations. [15] proposed combining the existence of different time scales with a heuristic gradient computation. However, the underlying assumptions and highly approximate gradient evaluation technique may place severe

limits on the applicability of this method.

5.2 Error Backpropagation Algorithm

The error backpropagation algorithm can often find a good set of weights in a reasonable amount of time. However, there are some drawbacks in most of the conventional error BP training clones:

(a) BP learning speed is sometimes very slow when the global minima are *well hidden* among local minima [11]. Slow speed may also come from its hill-climbing method for a minimum-seeking problem instead of optimization. Another conjecture for slow learning may arise from the fact that the error in the output layer is backpropagated to the hidden layers in the equivalent manner. In other words, this can mean that hidden and output layers use the same supervisory signal, which may be less reasonable in conjunction with the topology of multilayer neural networks. The derivation of *clear teaching signals* is expected to train the hidden layers reasonably.

(b) BP tends to get trapped at *local minima* since it is a method for calculating the gradient of the error with respect to the weight. With enough momentum, BP can escape these local minima. In this case, however, the next step weight estimates will not always be a better one.

(c) BP has a *scaling problem* such that its performance falls off rapidly on increased complexity networks even though it works well on simple training problems.

(d) BP's rate of convergence is seriously affected by *initial weights* and the *learning rate*. The development of a more robust method with respect to random initial values of synaptic weights is expected. The systematic or generalized selection of the learning rate is required for the various applications. BP can result in divergence and continuous instability if the learning step size is too large.

Many other researchers have proposed modifications of the classical BP algorithm to improve the drawbacks of the BP algorithm. [24] incorporates several heuristics laws in the BP algorithm, but they are difficult to describe systematically. [21] incorporated an extended Kalman filtering to improve the standard steepest descent technique. However, computational complexity of this algorithm becomes intractable as the size of the ANN increases. Recently, [19] and [10] partitioned a nonlinear neuron into linear and nonlinear portions (which was proposed by [19]), and used the Kalman filtering techniques to improve the learning speed and the sensitivity of the initial weights. However, they may suffer from the sensitivity problem on learning rate and similar drawbacks of BP algorithm due to the use of BP techniques for the hidden layer training.

5.3 Application of RLS to ANNs Training

The undesirable nature of the BP algorithm as a neural net training method and its close relative, Least-Mean-Square (LMS) algorithm leads to the use of recursive least-squares (RLS) algorithm. As a parameter estimation method for linear models such as ARX (AutoRegressive eXogeneous[1] input) and ARMAX (AutoRegressive Moving Average eXogeneous input). The RLS algorithm has been attractive because of its fast speed of convergence. But the direct application of the RLS parameter estimation algorithm to the neural training is difficult because of:

- (a) its applicability only to linear system models having single-layer-like architecture due to the requirement of a clear supervision signal for all layers being trained.
- (b) numerical instability like *wind-up* phenomenon when the covariance matrix does not meet the condition of *persistent excitation*.

---

[1] Econometric term, sometimes called eXtra or eXternal

- (c) crucial parameter setting problem like *forgetting factor* inside the RLS technique.

- (d) the non-universal justification of the Gaussian zero-mean noise assumption on the model mismatch[2] error.

Therefore, an improved version of the RLS algorithm was developed and was applied to the multi-layer ANNs training. This leads to the derivation of the Modified RLS (MRLS) algorithm shown in the Section 5.4. Since the MRLS originated from RLS, which in turn originated from the Least-Squares (LS) algorithm, the following two sections introduce the LS and RLS algorithms in a simple manner.

5.3.1 Least-Squares (LS) Algorithm

We consider below the difference equation 5.1 for the linear parametric model.

$$y_j^{act}(k) = \mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j + e_j(k), \qquad j = 1, \cdots, p \tag{5.1}$$

where $y_j^{act}(k) \in \Re^{1\times 1}$ is the desired signal (the teaching signal in ANNs) from measurement of a real system, $\mathbf{x}_j(k-1) \in \Re^{n\times 1}$ is the input data vector (sometimes called the regression vector), $\mathbf{w}_j \in \Re^{n\times 1}$ is the parameter vector assumed to be time-invariant and unknown, $\mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j$ is the model output, $e_j(k) \in \Re^{1\times 1}$ is the residual (modeling) error between actual and model outputs, and p is the number of the outputs for a system. To estimate the unknown parameter $\mathbf{w}_j$, the following criterion function is formulated to minimize the modeling error $e_j(k)$:

$$\begin{aligned} J[\mathbf{w}_j(N)] &\triangleq \frac{1}{N}\sum_{k=1}^{N} \gamma_j^{N-k} \cdot e_j^2(k) \\ &= \frac{1}{N}\sum_{k=1}^{N} \gamma_j^{N-k} \cdot [y_j^{act}(k) - \mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j]^2 \end{aligned} \tag{5.2}$$

---

[2] Sometimes, this mismatch between the behavior of the real system and that expected from the model is called *uncertainty*

where N is the number of measurements and γ_j is a sequence of positive numbers. The inclusion of the coefficients γ_j^{N-k} in the criterion 5.2 allows us to give different weights to different observations in terms such as *exponential-weight constant forgetting factor* (EW-CFF). An optimal choice of γ_j can be related to the variance of the noise term $e_j(k)$. The criterion is *quadratic* with respect to $\mathbf{w}_j$, which forms a bowl-shaped (concave upward) quadratic error function or *performance surface* such as a *paraboloid* (a ***hyper-paraboloid if there are more than two weights***) [25]. This gets free from a wrangle of local minima. The analytical minimization solution is shown in equation 5.3 provided the inverse exists.

$$(5.3)\quad \hat{\mathbf{w}} = \left[\sum_{k=1}^{N} \gamma_j^{N-k} \cdot \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1)\right]^{-1} \cdot \left[\sum_{k=1}^{N} \gamma_j^{N-k} \cdot \mathbf{x}_j(k-1) \cdot y_j^{act}(k)\right]$$

Equation 5.3 is called a celebrated *least-squares* (LS) algorithm by least sum of squared errors method. The detailed derivation of LS algorithm is shown in Section 5.4. Since equation 5.3 is of batch-mode (off-line), it can be rewritten in a recursive (on-line) fashion leading to the Recursive Least Squares (RLS) algorithm. In the following section, the RLS algorithm is introduced and its features are analyzed.

5.3.2 Standard RLS (SRLS) Algorithm

In Section 5.3.1, the LS method is applied to determine the optimal parameters in a given linear difference equation model. In using equation 5.3, all measurements must be available before computing $\hat{\mathbf{w}}(N)$. Thus, the computations are performed off-line. Suppose now that a new measurement $y_j^{act}(N+1)$ and data vector $\mathbf{x}_j(N)$ are obtained, and the new estimates $\hat{\mathbf{w}}(N+1)$ must be computed on the basis of all known measurements. If the LS algorithm of equation 5.3 is used, the entire sequence of mathematical operations should be repeated. It is very inefficient in computation, difficult to apply to real-time processing, and requires huge memory capacity while the excursion keeps on going. Practically, off-line estimation algorithms are not suitable for time-varying systems where N goes to infinity. It is possible, however, to manipulate equation 5.3 into a difference form that requires much fewer mathematical operations. This leads to SRLS algorithm which will only update the last estimate. The SRLS algorithm of equations 5.4 and 5.5 is obtained from LS method of equation 5.3 by forcing it into a recursive form. The detailed processes of deriving the SRLS algorithm is shown in Section 5.4.

(5.4)

$$\begin{aligned} \therefore \hat{\mathbf{w}}_j(k) &= \hat{\mathbf{w}}_j(k-1) + \mathbf{P}_j(k) \cdot \mathbf{x}_j(k-1)\left[y_j^{act}(k) - \mathbf{x}_j^T(k-1) \cdot \hat{\mathbf{w}}_j(k-1)\right] \\ &= \hat{\mathbf{w}}_j(k-1) + \mathbf{g}_j(k-1) \cdot \left[y_j^{act}(k) - \mathbf{x}_j^T(k-1) \cdot \hat{\mathbf{w}}_j(k-1)\right] \end{aligned}$$

$$\therefore P_j(k) = \frac{1}{\gamma_j}\left[P_j(k-1) - \frac{P_j(k-1)\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)P_j(k-1)}{\gamma_j + \mathbf{x}_j^T(k-1)P_j(k-1)\mathbf{x}_j(k-1)}\right] \tag{5.5}$$

$\mathbf{P}_j \in \Re^{n \times n}$ in the equation 5.5 is called the *covariance matrix* which is symmetric. The *estimates updating law* of equation 5.4 can be explained by the following representation.

$$\begin{pmatrix} \text{New} \\ \text{Estimates} \end{pmatrix} = \begin{pmatrix} \text{Previous} \\ \text{Estimates} \end{pmatrix} + \begin{pmatrix} \text{Algorithm} \\ \text{Gain} \end{pmatrix} \cdot \begin{pmatrix} \text{A Priori} \\ \text{Estimation Error} \end{pmatrix}$$

As a parameter estimation method, the SRLS algorithm has been attractive because of fast speed of convergence. However, there are still some points that need to be improved in order to apply it to practical projects and the neural network training. The limitations on the SRLS algorithm are as follows:

- (a) The updating law of equation 5.4 for unknown parameters has been derived under the assumption that the noise term in equation 5.1 is *Gaussian zero-mean white noise* [9]. The expectation of errors is zero in the long run. This assumption generally works reasonably with certain applications. However, the early stage of estimation may not justify this assumption, especially with a non-optimal initial parameter values. Real-life application can seldom meet this conjecture, and thereby, instability can take place when SRLS is applied to various situations. The Modified RLS (MRLS) algorithm in this research is derived in order to remove the assumption of Gaussian zero-mean noise on the modeling error.

- (b) The covariance matrix $\mathbf{P}_j(k)$ should be prevented from going to either zero and infinity. The covariance matrix can gradually decay to a small value with $\gamma_j \approx 1$, and the algorithm does not retain its *alertness* or adaptivity [20]. Since the second term on the right side of equation 5.5 is always positive or zero, $\mathbf{P}_j(k)$ gets smaller and smaller as time progresses. The smaller $\mathbf{P}_j(k)$ means smaller algorithm gain $\mathbf{g}_j(k-1)$ in equation 5.4. It is clear that smaller algorithm gain will give smaller modification for the new estimates. On the other hand, the *divergence* of the covariance matrix $\mathbf{P}_j(k)$ can also take place under the incomplete or no *persistent excitation* (PE), which is leading to the *estimates wind-up*. For example, by the small input vector of $\mathbf{x}_j(k-1) \approx 0$ and the forgetting factor of $\gamma_j \ll 1$ in equation 5.5, the covariance matrix will grow quickly. Therefore, the **covariance matrix composed of input vector and the forgetting factor plays an important role to keep the whole algorithm alive**. To do this, a certain process is necessary for the *covariance matrix modification* or the *periodic resetting*.

- (c) The effective selection method of forgetting factor should also be devised in terms such as *exponential-weight variable forgetting factor* to improve the SRLS algorithm.

These topics are tackled in this chapter.

Analysis On Application Of SRLS Algorithm To ANNs Training

Application to nonlinear model: An ANN is a nonlinear system, which is characterized by the activation function in an artificial neuron (except for linear activation function such as ADALINE, MADALINE, etc.). Since the RLS algorithm can be applied only to the parametric linear model, an adequate linearization process is required on the nonlinear models or a linear observation is necessary.

The *Taylor series expansion* is frequently used for linearization techniques. The *linear observation* can be carried out to take effect of linearization by means of applying the inversion process of the activation function to both neural network output and its supervision signal. In other words, this is the process of dividing a neuron model into linear and nonlinear portions, and the linear portion is only observed for the purpose of the neural network linearization provided that an activation function is invertable. The same inversion procedure is applied to its supervision signal. Then both linearly-observed signals are compared for the training. The linear observation method is preferred because of its reduced computational load.

Multi-layer training: The output layer in the multi-layer ANNs can be trained by the SRLS estimation method because the output layer can take an explicit teaching signal from the measurement of an actual system under consideration. On the other hand, in order to train the hidden (middle) layers, clear and reasonable teaching signals need to be supplied. **This makes the direct application of SRLS to the multi-layer neural networks difficult because SRLS requires the clear teaching signal**. For this purpose, the error back-propagation algorithms have used the error gradient from the output layer to the hidden layers consecutively, which can cause the aforementioned side-effects on BP clones. In this research, clear teaching signals for the hidden layers are derived based on the L_2-norm optimization technique.

Selection of forgetting factor: The proper value of the forgetting factor plays an important role for overall estimation performance in the SRLS algorithm. But, the optimal selection of a constant forgetting factor (CFF) value is difficult. Further the CFF may not be robust for various situations that arise in applying the algorithm to different systems. For example, the SRLS algorithm may not track the fast time-varying parameters even though they are able to chase the slow time-varying parameters or vice versa. This is referred to as the *adaptability for both slow and fast time-varying parameters.* It is generally known that the choice of forgetting factor ($0 < \lambda \leq 1$) influences the trade-off between tracking ability and noise (error) sensitivity such that a smaller value of forgetting factor gives faster discounting or forgetting of old signals, but leads to a great sensitivity to contemporary situation such as noise, and the opposite is true for forgetting factor close to one. A CFF can be used frequently for a slow time-varying system under the constrained situation. A CFF, however, may not cope with a more dynamic system under the diverse situation. This naturally leads to use of the variable forgetting factor (VFF) to carry both the time-varying and the time-invariant systems. A VFF requires an automatic decision of the forgetting value on-line at each sampling instance depending on the situation. But the use of VFF may not always guarantee the better performance over CFF of a fixed value, because most of VFF algorithms are generally built based on stochastic assumptions and the pre-knowledge of the system under consideration. The stochastic assumption on the system cannot always be justified under the real-life situation, and the use of the pre-knowledge on a system may result in an *ad-hoc* algorithm for a specific system under consideration. Hence, the algorithmic generality will frequently fail for a wide range of systems. In this research, the exponential-weight VFF is proposed without making assumptions and using pre-knowledge on the dynamic systems being studied. The proposed VFF algorithm relies on only measurements from the plant, which avoids the *ad-hoc* feature for the selection of forgetting factors. The proposed algorithm for VFF should be able to cope with different plants without alteration.

Divergence of covariance matrix: If the regressive input vector $\mathbf{x}_j(k)$ is not rich, the covariance matrix of $\mathbf{P}_j(k)$ will grow exponentially, what is called the *burst phenomenon* in the covariance matrix. The required properties of the input vector are, in loose terms, related to the fact that the input should excite all modes of the system by keeping *alert* values. Such an input will be called *persistent excitation* (PE), or *general enough* or *rich enough*. The requirement of PE, therefore, means that the input must have sufficiently rich frequency content. For example, in the extreme case of $\mathbf{x}_j(k) \approx 0$, $\mathbf{P}_j(k) = \mathbf{P}_j(k-1)/\lambda$ goes to ∞ for $0 < \lambda < 1$. Therefore, the covariance matrix should be suppressed in cases of the incomplete excitation without seriously losing the original information on the plant being controlled if possible, or the estimates updating law must have enough robustness to recover the lost information that occurs in the process of suppressing the covariance matrix divergence. This is a difficult task and is an active research topic. In this thesis, the periodical resetting method together with the EW-VFF is used without degrading the performance, and guarantees suppressing of the burst phenomenon at least for the systems studied in this thesis.

Sensitivity on initial parameters: The poor selection of initial parameter values being estimated causes serious convergence problems in the SRLS clones, and frequently falls into the overall *estimates wind-up*. Other gradient algorithms including BP and LMS are also prone to the sensitivity problem of initialization. In this research, this problem is tackled by using of the Modified Recursive Least-Square (MRLS) algorithm, which is based on different *assumptions on modeling error* from that of the SRLS algorithm. The MRLS algorithm appears to be less sensitive with respect to the selection of initial parameter values because it avoids zero-mean noise which is assumed to be a zero after summing up. This assumption may not be true for the various systems, which can result in biased estimates or failure of estimation. Especially, at the beginning phase of the training, the side-effect of zero-mean noise assumption can become more serious with the non-optimal initial parameters which produces naturally larger initial modeling error.

5.4 Modified RLS (MRLS) Algorithm

In this section, the SRLS algorithm is modified for the MRLS algorithm. We consider the same difference equation 5.6 for the linear parametric model.

$$y_j^{act}(k) = \mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j(k) + e_j(k), \qquad j = 1, \cdots, p \tag{5.6}$$

5.4.1 Derivation of Estimates Update Law

The cost function of equation 5.7 is minimized at first to derive the LS algorithm batch mode with an ***exponential-weight variable forgetting factor*** (EW-VFF) as shown in equation 5.8.

(5.7)

$$\begin{aligned} J[\mathbf{w}_j(N)] &\triangleq \frac{1}{N}\sum_{k=1}^{N} \gamma_j^{N-k}(k) e_j^2(k) \\ &= \frac{1}{N}\left\{\gamma_j^{N-1}(1)e_j^2(1) + \gamma_j^{N-2}(2)e_j^2(2) + \cdots + \gamma_j^1(N-1)e_j^2(N-1) + 1 \cdot e_j^2(N)\right\} \\ &\approxeq \sum_{k=1}^{N} \gamma_j^{N-k}(k)[y_j^{act}(k) - \mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j(N)]^2 \end{aligned}$$

where $0 < \gamma_j(k) < 1$. $\frac{\partial J}{\partial \mathbf{w}_j(N)} = 0$ gives the following,

$$\sum_{k=1}^{N}\left\{-\mathbf{x}_j(k-1) \cdot \gamma_j^{N-k}(k) \cdot \left[y_j^{act}(k) - \mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j(N)\right]\right\} = 0$$

(5.8)

$$\hat{\mathbf{w}}(N) = \left[\sum_{k=1}^{N} \gamma^{N-k}(k) \cdot \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1)\right]^{-1} \cdot \left[\sum_{k=1}^{N} \gamma^{N-k}(k) \cdot \mathbf{x}_j(k-1) \cdot y_j^{act}(k)\right]$$

In equation 5.8, the inside of the left bracket is defined as equation 5.9.

$$R(N) \triangleq \sum_{k=1}^{N} [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1) \tag{5.9}$$

and

$$(5.10)\qquad R(N-1) \triangleq \sum_{k=1}^{N-1} [\gamma_j(k)]^{N-1-k} \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1)$$

where $R(N)$ is a symmetrical matrix which gives the *covariance matrix* symmetry, which will be detailed later. Equation 5.8 is replaced by equation 5.9 to get equation 5.11.

(5.11)

$$\begin{aligned}
\hat{\mathbf{w}}_j(N) &= \left[R(N)\right]^{-1} \cdot \left[\sum_{k=1}^{N} [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1) \cdot y_j^{act}(k)\right] \\
&= \left[R(N)\right]^{-1} \cdot \left[\sum_{k=1}^{N-1} \left\{ [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1) \cdot \underbrace{y_j^{act}(k)}_{\text{see eq5.12}} \right\} + [\gamma_j(N)]^0 \mathbf{x}_j(N-1) y_j^{act}(N)\right] \\
&\approxeq\rightarrow
\end{aligned}$$

where $\sum_{k=1}^{N-1} y_j^{act}(k)$ is replaced as shown below.

Removal of Zero-mean Assumption on Modeling Error

In equation 5.11, the expectation of the desired signal $\sum_{k=1}^{N-1} y_j^{act}(k)$ is rewritten as equation 5.12, which is different from the SRLS algorithm where $\sum_{k=1}^{N-1} e_j(k) = 0$ is assumed based on *Gaussian zero-mean white noise* on the residual error.

$$
\begin{aligned}
\sum_{k=1}^{N-1} y_j^{act}(k) &= \sum_{k=1}^{N-1}\left\{\mathbf{x}_j^T(k-1)\hat{\mathbf{w}}_j(k) + e_j(k)\right\} \\
&= \Big\{[\mathbf{x}_j^T(0)\hat{\mathbf{w}}_j(1) + e_j(1)] + [\mathbf{x}_j^T(1)\hat{\mathbf{w}}_j(2) + e_j(2)] + \cdots \\
&\qquad \cdots + [\mathbf{x}_j^T(N-2)\hat{\mathbf{w}}_j(N-1) + e_j(N-1)]\Big\} \\
&\approxeq \Big\{[\mathbf{x}_j^T(0)\hat{\mathbf{w}}_j(N-1)] + [\mathbf{x}_j^T(1)\hat{\mathbf{w}}_j(N-1)] + \cdots + [\mathbf{x}_j^T(N-2)\hat{\mathbf{w}}_j(N-1) \\
&\qquad + [e_j(1) + e_j(2) + \cdots + e_j(N-1)]\Big\} \\
&= \sum_{k=1}^{N-1}\left\{\mathbf{x}_j^T(k-1)\hat{\mathbf{w}}_j(N-1) + e_j(k)\right\} \\
&\approxeq \sum_{k=1}^{N-1}\left\{\mathbf{x}_j^T(k-1)\hat{\mathbf{w}}_j(N-1) + e_j(N-1)\right\}
\end{aligned}
\tag{5.12}
$$

If $\sum_{k=1}^{N-1} e_j(k) = 0$ is assumed as SRLS algorithm, the approximate equal sign of $\approxeq$ in equation 5.12 becomes much less reasonable especially when N is small. Small N represents the beginning phase of the estimation where the modeling errors are generally large. However, equation 5.12 for the MRLS algorithm gets rid of the assumption that the noise terms would disappear just by adding them up as time progresses in SRLS algorithm. In equation 5.12, the use of the approximate equal sign of $\approxeq$ is accompanied by the error term, which can be a natural notion leading to the overall algorithmic robustness.

$$\begin{aligned}
\rightarrow \approxeq & \Big[R(N)\Big]^{-1} \cdot \Big[\sum_{k=1}^{N-1}\Big\{[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\Big\{\mathbf{x}^T(k-1)\hat{\mathbf{w}}_j(N-1)+e_j(N-1)\Big\}\Big\} \\
& + \mathbf{x}(N-1)y_j^{act}(N)\Big] \\
= & \Big[R_j(N)\Big]^{-1} \cdot \Big[\sum_{k=1}^{N-1}\Big\{[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)\hat{\mathbf{w}}_j(N-1)\Big\} + \\
& \qquad \sum_{k=1}^{N-1}\Big\{[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)e_j(N-1)\Big\} + \mathbf{x}_j(N-1)y_j^{act}(N)\Big]
\end{aligned}$$

(5.13)

$$\begin{aligned}
= & \Big[R(N)\Big]^{-1} \cdot \Big[\underbrace{\Big\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)\Big\}}_{\text{(a) see eq.5.15}}\hat{\mathbf{w}}_j(N-1) \\
& \qquad + \underbrace{\Big\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\Big\}}_{\text{(b) see eq.5.16}}e_j(N-1) + \mathbf{x}_j(N-1)y_j^{act}(N)\Big]
\end{aligned}$$

$$= \rightarrow$$

(a) in equation 5.13 is written into equation 5.15 using the earlier definition of equation 5.9.

(5.14)

$$\begin{aligned}
R_j(N) \triangleq & \sum_{k=1}^{N}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1) \\
= & \Big\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)\Big\} + [\gamma_j(k)]^0\mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)
\end{aligned}$$

$$(5.15) \quad \therefore \quad \sum_{k=1}^{N-1}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1) = R_j(N) - \mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)$$

(b) in equation 5.13 is written into equation 5.16 using below new definition:

$$Q_j(N) \triangleq \sum_{k=1}^{N} [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1)$$

$$\Downarrow$$

$$Q_j(N-1) \triangleq \sum_{k=1}^{N-1} [\gamma_j(k)]^{N-1-k} \mathbf{x}_j(k-1)$$

$$\left\{\sum_{k=1}^{N-1} [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1)\right\} \approxeq \gamma_j(N-1) \cdot \sum_{k=1}^{N-1} \left\{[\gamma_j(k)]^{N-1-k} \mathbf{x}_j(k-1)\right\}$$

$$= \gamma_j(N-1) \cdot Q_j(N-1) \tag{5.16}$$

By substituting equations 5.15 and 5.16 into (a) and (b) in equation 5.13 respectively, yields,

(5.17)

$$\begin{aligned}
\rightarrow = & \left[R_j^{-1}(N)\right] \Big[\left\{R_j(N) - \mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)\right\} \hat{\mathbf{w}}_j(N-1) + \\
& \qquad \left\{\gamma_j(N-1) \cdot Q_j(N-1)\right\} e_j(N-1) + \mathbf{x}_j(N-1) y_j^{act}(N) \Big] \\
= & \underbrace{R_j^{-1}(N) R_j(N)}_{=[\mathbf{I}]} \hat{\mathbf{w}}_j(N-1) - R_j^{-1}(N) \mathbf{x}_j(N-1) \mathbf{x}_j^T(N-1) \hat{\mathbf{w}}_j(N-1) \\
& + R_j^{-1}(N) \gamma_j(N-1) Q_j(N-1) e_j(N-1) + R_j^{-1}(N) \mathbf{x}_j(N-1) y^{act}(N) \\
= & \hat{\mathbf{w}}_j(N-1) + R_j^{-1}(N) \cdot \mathbf{x}_j(N-1) \cdot \left\{y_j^{mact}(N) - \mathbf{x}_j^T(N-1) \hat{\mathbf{w}}_j(N-1)\right\} \\
& + R_j^{-1}(N) \cdot \gamma_j(N-1) \cdot Q_j(N-1) \cdot e_j(N-1)
\end{aligned}$$

where the error term is replaced by equation 5.18 which in turn is based on equation 5.6.

$$e_j(N-1) = y_j^{act}(N-1) - \mathbf{x}_j^T(N-2) \hat{\mathbf{w}}_j(N-1), \tag{5.18}$$

Therefore, the parameter ***estimates update law*** for the MRLS is shown as:

(5.19)

$$\begin{aligned}\therefore \hat{\mathbf{w}}_j(N) = \hat{\mathbf{w}}_j(N-1) &+ R_j^{-1}(N)\mathbf{x}_j(N-1)\Big\{y_j^{act}(N) - \mathbf{x}_j^T(N-1)\hat{\mathbf{w}}_j(N-1)\Big\} \\ &+ R_j^{-1}(N)\gamma_j(N-1)Q_j(N-1)\Big\{y_j^{act}(N-1) - \mathbf{x}_j^T(N-2)\hat{\mathbf{w}}_j(N-1)\Big\}\end{aligned}$$

The estimates update law of equation 5.19 for the MRLS algorithm is shown to be different for the SRLS algorithm. Now the *matrix inversion lemma* is applied since equation 5.19 contains the matrix inversion process. This leads to the *covariance matrix update law.*

5.4.2 Derivation of Covariance Matrix Update Law

Using Matrix Inversion Lemma

Since the matrix $R_j(N)$ in equation 5.19 should be inverted in each updating step, the matrix inversion lemma is used to avoid repeated matrix inversions. The matrix inversion lemma of equation 5.20 is applied to the $R_j^{-1}(N)$ based on its definition of equation 5.14 as follows:

(5.20)

$$\begin{aligned}R_j^{-1}(N) &= \left[\sum_{k=1}^{N}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)\right]^{-1} \\ &= \left[\left\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)\right\}+\mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)\right]^{-1} \\ &\approxeq \left[\gamma_j(N-1)\underbrace{\left\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-1-k}\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)\right\}}_{\text{see eq.5.15}} + \mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)\right]^{-1} \\ &= \left[\gamma_j(N-1)\cdot R_j(N-1) + \mathbf{x}_j(N-1)\cdot\mathbf{x}_j^T(N-1)\right]^{-1} \\ &= \rightarrow\end{aligned}$$

where $\gamma_j(N-1)$ is the scalar forgetting factor. Below is the general *matrix inversion lemma.*

$$R(N) \triangleq \sum_{k=1}^{N} [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1)$$

$$\Downarrow$$

$$R(N-1) \triangleq \sum_{k=1}^{N-1} [\gamma_j(k)]^{N-1-k} \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1)$$

$$[a \cdot A + BC]^{-1} = \frac{A^{-1}}{a} - \left[\frac{\frac{A^{-1}}{a} BC \frac{A^{-1}}{a}}{1 + C\frac{A^{-1}}{a} B}\right], \qquad [a \cdot A]^{-1} = \frac{A^{-1}}{a}$$

where a is a scalar.

$$\begin{aligned}
\rightarrow &= \Big[\gamma_j(N-1)R_j(N-1)\Big]^{-1} - \\
&\left[\frac{[\gamma_j(N-1)R_j(N-1)]^{-1}\mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)[\gamma_j(N-1)R_j(N-1)]^{-1}}{1 + \mathbf{x}_j(N-1)[\gamma_j(N-1)R_j(N-1)]^{-1}\mathbf{x}_j^T(N-1)}\right] \\
&= \frac{1}{\gamma_j(N-1)}\left[R_j^{-1}(N-1) - \frac{R_j^{-1}(N-1)\mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)R_j^{-1}(N-1)}{\gamma_j(N-1) + \mathbf{x}_j^T(N-1)R_j^{-1}(N-1)\mathbf{x}_j(N-1)}\right]
\end{aligned}$$

(5.21)

$$R_j^{-1}(N) = \frac{1}{\gamma_j(N-1)}\left[R_j^{-1}(N-1) - \frac{R_j^{-1}(N-1)\mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)R_j^{-1}(N-1)}{\gamma_j(N-1) + \mathbf{x}_j^T(N-1)R_j^{-1}(N-1)\mathbf{x}_j(N-1)}\right]$$

The definition of equation 5.22 below is applied to equation 5.21 resulting in equation 5.23, where P_j is called the *covariance matrix*[3] and is a symmetric matrix.

(5.22)

$$\begin{aligned}
P_j(N) &\triangleq R_j^{-1}(N) \\
&= \left[\sum_{k=1}^{N} [\gamma_j(k)]^{N-k} \mathbf{x}_j(k-1) \cdot \mathbf{x}_j^T(k-1)\right]^{-1}
\end{aligned}$$

---

[3] [5] shows that the expectation of $P_j(K)$ is proportional to the covariance matrix of the parameter estimates when the model equation 5.6 is a true linear regression. For this reason, $P_j(K)$ is usually referred to as the *covariance matrix*.

(5.23)

$$\therefore P_j(N) = \frac{1}{\gamma_j(N-1)}\left[P_j(N-1) - \frac{P_j(N-1)\mathbf{x}_j(N-1)\mathbf{x}_j^T(N-1)P_j(N-1)}{\gamma_j(N-1) + \mathbf{x}_j^T(N-1)P_j(N-1)\mathbf{x}_j(N-1)}\right]$$

Equation 5.23 is called the *covariance matrix update law* with *exponential-weight variable forgetting factor*. In equation 5.23, one should restrict attention to the input vector $\mathbf{x}_j(k)$ as to whether it satisfies the *persistent excitation* (PE) condition shown below,

(5.24)
$$\alpha \cdot \mathbf{I} \leq R_j(k) \leq \beta \cdot \mathbf{I}$$

where the positive constants α and β exist strictly for sufficiently large k. When the PE condition is not satisfied, the covariance matrix in equation 5.23 can approach zero or infinity, which leads the estimates update law to *turn-off* or *wind-up*. Therefore, special attention to the covariance matrix update law is required depending on the input regression vector $\mathbf{x}_j(k)$ in order to avoid numerical problems. This topic will be dealt with in Section 5.4.7.

5.4.3 Derivation of Input Vector Update Law

In equation 5.19 for the parameter updating law, $Q_j(N)$ vector is defined in non-recursive form in equation 5.19 which is rewritten into the recursive form as follows:

$$\begin{aligned}
Q_j(N) &\triangleq \sum_{k=1}^{N}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1) \\
&= \left\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-k}\mathbf{x}_j(k-1)\right\} + \mathbf{x}_j(N-1) \\
&\cong \gamma_j(N-1)\underbrace{\left\{\sum_{k=1}^{N-1}[\gamma_j(k)]^{N-1-k}\mathbf{x}_j(k-1)\right\}}_{\downarrow} + \mathbf{x}_j(N-1) \\
&= \gamma_j(N-1)\cdot\overbrace{Q_j(N-1)}^{\downarrow} + \mathbf{x}_j(N-1)
\end{aligned}$$

$$\therefore \quad Q_j(N) = \gamma_j(N-1) \cdot Q_j(N-1) + \mathbf{x}_j(N-1) \tag{5.25}$$

Equation 5.25 is called the ***input vector update law*** in this research.

5.4.4 Summary of MRLS Algorithm

In equations 5.19, 5.23 and 5.25, the dummy variable N is replaced by the time sequence variable k, which results in equations 5.26, 5.27 and 5.28 composed of ***estimates update law***, ***covariance update law*** and ***input vector update law*** respectively. These are called ***Modified Recursive Least-Squares*** (MRLS) algorithm with ***Exponential-Weight Variable Forgetting Factor*** (EW-VFF) in this research. These are shown as follows:

(I) Estimates Update Law:

(5.26)

$$\begin{aligned} \therefore \hat{\mathbf{w}}_j(k) = \hat{\mathbf{w}}_j(k-1) &+ P_j(k) \cdot \mathbf{x}_j(k-1) \cdot \left\{ y_j^{act}(k) - \mathbf{x}_j^T(k-1) \cdot \mathbf{w}_j(k-1) \right\} \\ &+ P_j(k) \cdot \gamma_j(k-1) \cdot Q_j(k-1) \cdot \left\{ y_j^{act}(k-1) - \mathbf{x}_j^T(k-2) \cdot \hat{\mathbf{w}}_j(k-2) \right\} \end{aligned}$$

(II) Covariance Matrix Update Law:

(5.27)

$$\therefore P_j(k) = \frac{1}{\gamma_j(k-1)} \left[P_j(k-1) - \frac{P_j(k-1)\mathbf{x}_j(k-1)\mathbf{x}_j^T(k-1)P_j(k-1)}{\gamma_j(k-1) + \mathbf{x}_j^T(k-1)P_j(k-1)\mathbf{x}_j(k-1)} \right]$$

(III) Input Vector Update Law:

$$\therefore Q_j(k) = \gamma_j(k-1) \cdot Q_j(k-1) + \mathbf{x}_j(k-1) \tag{5.28}$$

5.4.5 Expected Features Of MRLS Algorithm

The MRLS algorithm has created a new polynomial term (the third term in the right-hand side) and generated the ***input vector update law*** as shown in equations 5.26 and 5.28 respectively in contrast to the SRLS algorithm. The new term in the equation 5.26 has been produced by removing the zero-mean assumption on the modeling error in the process of its derivation. This new term is expected to smooth the weight changes by ***over-relaxation***, that is, by adding a kind of momentum term as does in the BP algorithm. By chance, this novel term resembles the momentum term in the BP algorithm although the original motivation is totally different. This addition may improve the convergence rate, steady state performance and robustness together with the developed EW-VFF (exponential-weight variable forgetting factor) under a wide range of operating conditions. The important difference is that the new term in the MRLS is automatically computed at every step by the algorithm, while the momentum rate α in the BP algorithm [16] has to be guessed by the user (typical value, $\alpha = 0.9$). It is well known that a momentum term is useful for not only on-line learning but also for a batch learning algorithm. Chapter 6 shows excellent performance of the MRLS algorithm over the SRLS algorithm.

5.4.6 Design Of New Exponential-Weight Variable Forgetting Factor

General Features Of the Forgetting Factor

An important requirement of the SRLS algorithm for adaptive control and adaptive signal processing lies in their ability to track parameter changes. In the SRLS algorithm with ***exponential-weight constant forgetting factor*** (EW-CFF) of γ_j^{N-k} for $0 < \gamma_j < 1$, past measurements are discounted and the algorithm is forced to concentrate on more recent data. The choice of forgetting factor γ_j value becomes a very important issue because a $\gamma_j \ll 1$ gives fast tracking while a $\gamma_j \rightarrow 1$ sluggish tracking ability. However, the lower the value of the forgetting factor, the higher the tracking velocity but more sensitivity of the noise. This motivates the use of *exponential-weight variable forgetting factor* (EW-VFF) of $\gamma_j^{N-k}(k)$ instead of constant γ_j^{N-k}. However, it is difficult to determine automatically whether the parameter estimation requires fast or slow parameter tracking during implementation. In this author's opinion, the necessary properties of a VFF should be:

1. Case (I): At the beginning of the estimation, an abrupt change of parameter is expected due to the poor choice of the unknown initial estimates, and the modeling error becomes large. In this phase $\gamma(k_0) \ll 1$ is desirable. To handle this situation, [8] proposed the exponentially growing forgetting factor depending on the time k such that it forgets data during only the beginning phase by a smaller γ_j and then let it go back to $\gamma \to 1$ as the effect of initial condition diminishes such that:

   $$\gamma(k) = \gamma(0) \cdot \gamma(k-1) + [1 - \gamma(0)] \tag{5.29}$$

   where the initial values are $\gamma(0) = 0.95 \sim 0.98$. The natural interpretation of equation 5.29 is that it simply grows from $\gamma(0)$ to 1 as time progresses, $k = 1, 2, 3, \ldots$. Therefore, this forgetting factor may not adaptively work for abrupt parameter changes after the initial estimation phase is over. This method cannot detect abrupt parameter changes in the middle of excursion.

2. Case (II): After the effect of the initial condition diminishes, the real parameters in plant dynamics can be changed abruptly by the uncertainty in plant dynamics: for example, when a robot arm picks up or releases the unknown payload at any time, which causes quick change of link mass; when a fuel tank of the rocket is released; and when unpredictable fluid dynamics change occurs to an underwater vehicle. In these situations, the small forgetting factor of $\gamma(k) \ll 1$ is also required. To handle these situations, [13] devised a novel variable forgetting factor such that when the parameters are changed abruptly, the forgetting factor is kept small, and when the estimates vector converges to the true value, then $\gamma(k)$ is increased to unity. This is performed by equation 5.30.

$$\begin{aligned} \gamma(k) &= \gamma_{min} + (1 - \gamma_{min}) \cdot 2^{L(k)} \\ L(k) &= -NINT[\rho \cdot \alpha^2(k)]] \\ &= -NINT\left[\rho \cdot \left(y^{act} - \mathbf{x}^T(k-1) \cdot \hat{\mathbf{w}}(k)\right)^2\right] \end{aligned} \tag{5.30}$$

where $NINT[\cdot]$ is defined as the nearest integer to $[\cdot]$ by means of a rounding operation, $\rho = 5$ called the *sensitivity gain* and the minimum VFF of $\gamma_{min} = 0.7$. This is interesting in that situation for the true parameter value is recognized in terms of the estimation error $\alpha(k)$ in equation 5.30. However, it is unclear whether it will work for continuously time-varying parameters due to its *binary-like* decision rule in the algorithm, and whether the adjustable design constants can be generalized for different plants. This research did not show the simulation for a plant of continuously-changing system parameters.

3. Case (III): When a plant of continuously time-varying parameters is supposed to track the profile of the variable desired signal, the role of the variable forgetting factor becomes more serious than in case (I) and (II). A certain time-varying system to be controlled can be categorized into two cases: (a) A plant of time-varying coefficients, where its coefficients are not dependent on the output states, does not create a serious difference between the *regulator* (a matter of tracking the constant desired signal on a time-varying system) and the *tracker* (a matter of tracking time-varying desired signal on a time-varying system). (b) A plant of time-varying coefficients, where its coefficients are also a function of the output states, does make a difference between the regulator and tracker because the tracker requires a more robust parameter estimation algorithm than the regulator does (These are different stories from the general regulating and tracking problems for a time-invariant system). A typical example of the tracking problem in (b) is a robot manipulator that is supposed to pick up a fast-moving/heavy target and release it back along the desired path. This case is a combination of an abrupt change plus a continuously time-variance in the

system parameters. This example motivates the development of a robust and flexible variable forgetting factor.

Although the above cases present real-life situations, the task of how to develop a remedy for situations involving fast, slow and/or constant variation in system parameters presents complex problems. Other research on the variable forgetting factor in conjunction with RLS algorithm are shown as follows: Adaptive Forgetting Through Multiple Models (AFMM) algorithm was introduced by [1]. Its ability to track sudden parameter changes can be good. The main drawback of the method is the use of a Gaussian sum approximation in the algorithm and its heavy computation demand since it requires multiple recursive least squares (RLS) algorithm. Besides, the estimates of the unchanged parameters can be badly disturbed. It was tested on a simple SISO system of only one time-varying parameter. [4] proposed a tracking problem which uses a UD factorization method in updating the covariance matrix instead of updating itself to detect the information on parameter changes. For instance, updating the covariance matrix is suspended by modifying the recursive gain matrix of a RLS algorithm and putting the effect of forgetting factor to unity. This algorithm is simpler than the AFMM algorithm. The disadvantage of this method is a delay of several steps in detecting a parameter change, and this may cause a large fluctuation in parameter estimates when a false alarm occurs. In general, a false alarm is unavoidable. [23] improved the algorithm of [4] by forming more robust detection signals for parameter changes. The method is considered for application to a MIMO system. This RLS estimation algorithm uses a constant forgetting factor, but dead zones are introduced to the recursive parameter update part under binary criterion to determine the time of updating for constant parameters. A threshold value for a detecting signal should be determined. This algorithm may have the possibility of a false alarm and may not work for a system with continuously changing (updating) parameters since it uses a constant forgetting factor in the parameter updating phase. This was tested on a system having both constant parameters and short-range variable and long-range constant parameters in [23]. [13] presented a new exponential-weight variable forgetting factor for the SRLS algorithm as shown in equation 5.30. The presented method has good tracking adaptability with a small forgetting factor in the nonstationary situation (i.e., at the beginning of estimation and in the situation of sudden parameters change in the middle of estimation) and with a unity in the stationary environments (i.e., in the situation of constant parameters). This algorithm was not tested for a continuously time-varying system with an abrupt parameter change, and did not show the generality of the adjustable design parameter for different systems to justify the *non ad-hoc* algorithm.

Design Of A New Variable Forgetting Factor

The new EW-VFF developed in this thesis meets two design criteria: (a) The VFF algorithm avoids the *prior* knowledge and assumptions on systems under consideration. This objective makes a more generalized FF for various systems possible. This kind of FF can be built based on only measurements from the concerned systems. (b) The VFF clearly indicates the methodology to determine the real VFF value for on-line (real-time) applications. With rough range for the VFF, it is difficult to determine the precise value for the on-line application which requires the continuous update of VFF at each sampling instance. Then it needs to illustrate the time-history profile of the VFF.

To carry out the above objectives in this research, the use of *Cauchy function* is proposed as shown in equation 5.31 and Figure 34, where $x_j(k)$ depends on two signals of $y_j^{act}(k)$ and $y_j^{act}(k-1)$.

$$\gamma_j(k) = \frac{c}{c + [x_j(k)]^2} \tag{5.31}$$

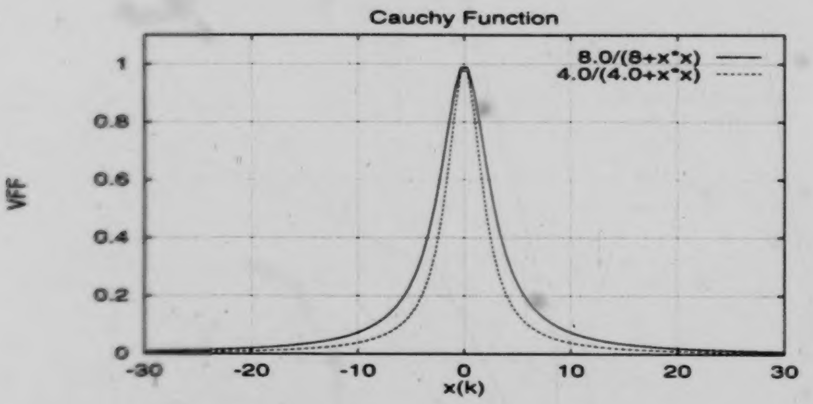


Figure 34: Cauchy Function Profile of $x_j(k)$

In Figure 34, the Y-axis represents the magnitude of $\gamma_j(k)$ which always stays in the range of $0 < \gamma_j(k) \leq 1$ regardless of the *Cauchy constant* c value. This meets one of the requirements for the forgetting factor. Here the constant just determines the envelope sharpness of the Cauchy function such that as c gets smaller, the Cauchy function envelop gets sharper. On the other hand, the X-axis for the variable $x_j(k)$ has the dynamic range such that a visual interpretation of Figure 34 represents (i) that $x_j(k) \rightarrow 0$ makes $\gamma_j(k) \rightarrow 1$ (represents the situation of constant or slowly time-varying systems), and (ii)that $\mid \pm x_j(k) \mid$ of far-from-zero forces $\gamma_j(k) \ll 1$ (implies the situation of (fast) time-varying systems). Therefore, the algorithm for determining the dynamic variable $x_j(k)$ plays a crucial role for the new variable forgetting factor $\gamma_j(K)$. In equation 5.31, the Cauchy constant has been chosen for $c = 4.0 \sim 8.0$, and $\gamma_j(k)$ is forced to 0.7 for less than 0.7 empirically. But it does not lose the generality for a variable forgetting factor because the time-variable property of $\gamma_j(k)$ is given by the variable $x_j(k)$. Later computer simulation shows good performance with an almost-fixed $c = 5.0$ and variable $x_j(k)$ for different systems.

Now the algorithm to determine $x_j(k)$ is developed based on the hypothesis that the main information about fast/slow parameter tracking situation can come from the measurements of system outputs. Let $y_j^{act}(k)$ and $y_j^{act}(k-1)$ be the present and the previous actual measurements respectively at time instance k. The ratio of

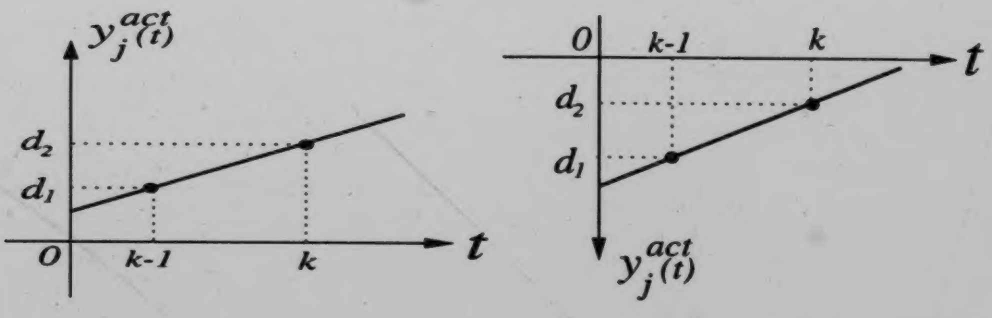
$y_j^{act}(k)/y_j^{act}(k-1)$ determines $x_j(k)$ for the self-adjusting forgetting factor as shown in equation 5.32 for a simple case.

$$\begin{aligned} \gamma_j(k) &= \frac{c}{c+[x_j(k)]^2} \\ &= \frac{c}{c+[\frac{y_j^{act}(k)}{y_j^{act}(k-1)}]^2} \end{aligned} \tag{5.32}$$

The philosophy behind equation 5.32 is explained such that the larger output change-ratio of $y_j^{act}(k)/y_j^{act}(k-1)$ would imply that more change for the time-varying parameters is required and that vice versa could be justified as well. The detailed description is examined by the following graphics and computer pseudo-codes depending on all possible patterns for both $y_j^{act}(k)$ and $y_j^{act}(k-1)$ of outputs measurements.

Graphical Illustration For $\mathbf{x}_j(k)$

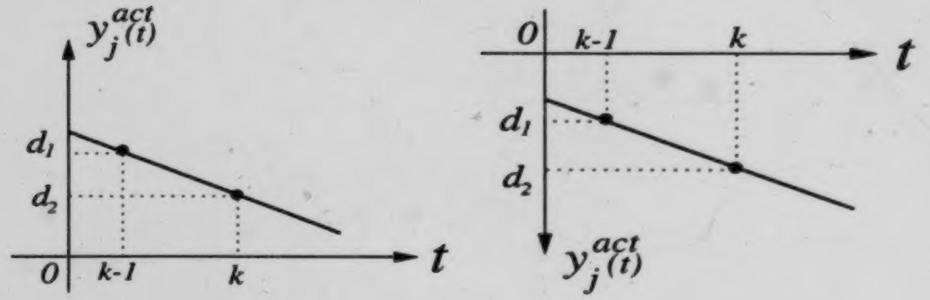
(I) In the case of $y_j^{act}(k-1) \cdot y_j^{act}(k) > 0$



(a) When both positive, and $d_1 < d_2$

(b) When both negative, and $d_1 < d_2$

Figure 35: Then $(a): x_j(k) = \frac{y_j^{act}(k)}{y_j^{act}(k-1)}$, $(b): x_j(k) = \frac{y_j^{act}(k-1)}{y_j^{act}(k)}$

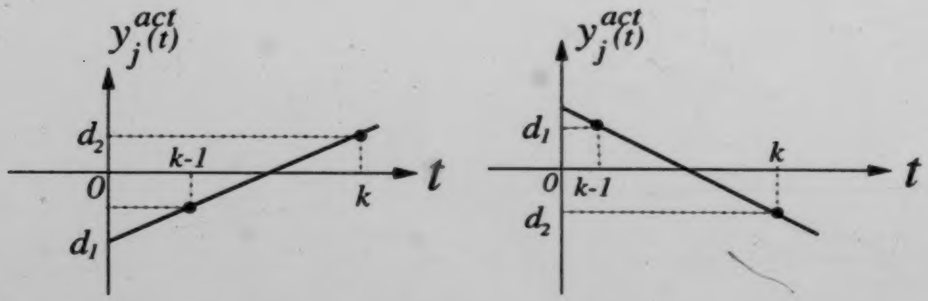

$y_j^{act}(t)$
d_1
d_2
0
k-1
k
t
0
k-1
k
t
d_1
d_2
$y_j^{act}(t)$

(a) When both positive, and $d_1 > d_2$

(b) When both negative, and $d_1 > d_2$

Figure 36: Then $(c):\ x_j(k) = \frac{y_j^{act}(k-1)}{y_j^{act}(k)}$, $(d):\ x_j(k) = \frac{y_j^{act}(k)}{y_j^{act}(k-1)}$

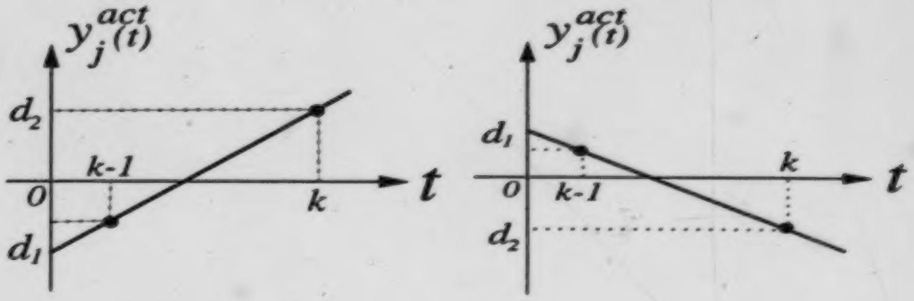
(II) In the case of $y_j^{act}(k-1) \cdot y_j^{act}(k) < 0$


$y_j^{act}(t)$
d_2
k-1
0
k
t
d_1
$y_j^{act}(t)$
d_1
k
0
k-1
t
d_2

(a) When d_1 negative, d_2 positive, and same magnitude

(b) When d_1 positive, d_2 negative, and same magnitude

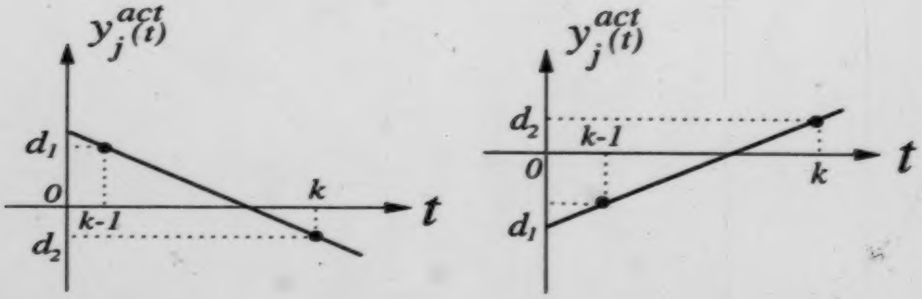
Figure 37: Then $(e):\ x_j(k) = \frac{[|y_j^{act}(k)|-|y_j^{act}(k-1)|]+y_j^{act}(k)}{y_j^{act}(k)} = 1$, $(f):\ x_j(k) = \frac{[|y_j^{act}(k-1)|-|y_j^{act}(k)|]+y_j^{act}(k-1)}{y_j^{act}(k-1)} = 1$


$y_j^{act}(t)$
d_2
k-1
0
k
t
d_1
$y_j^{act}(t)$
d_1
k
t
0
k-1
d_2

(a) When d_1 negative, d_2 positive, and d_1 magnitude smaller than d_2 magnitude

(b) When d_1 positive, d_2 negative, and d_1 magnitude smaller than d_2 magnitude

Figure 38: Then $(g)\&(h):\ x_j(k) = \frac{[|y_j^{act}(k)| - |y_j^{act}(k-1)|] + |y_j^{act}(k-1)|}{w} y_j^{act}(k-1) = \frac{y_j^{act}(k)}{y_j^{act}(k-1)}$

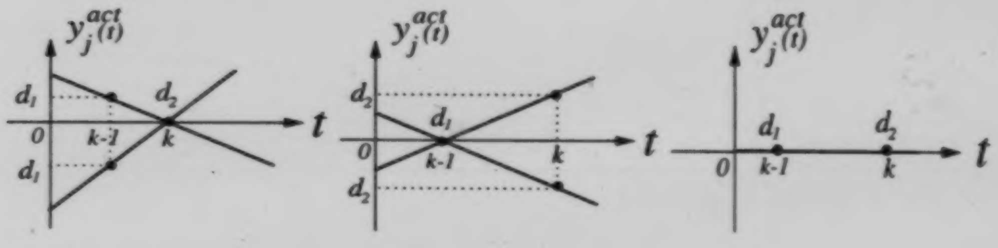

$y_j^{act}(t)$
d_1
0
k
t
k-1
d_2
$y_j^{act}(t)$
d_2
k-1
t
0
k
d_1

(a) When d_1 positive, d_2 negative, and d_1 magnitude greater than d_2 magnitude

(b) When d_1 negative, d_2 positive, and d_1 magnitude greater than d_2 magnitude

Figure 39: Then $(i)\&(j):\ x_j(k) = \frac{[|y_j^{act}(k-1)| - |y_j^{act}(k)|] + |y_j^{act}(k)|}{y_j^{act}(k)} = \frac{y_j^{act}(k-1)}{y_j^{act}(k)}$

(III) In the case of $y_j^{act}(k-1)\cdot y_j^{act}(k)=0$


$y_j^{act}(t)$
d_1
d_2
0
k-1
k
d_1
t
$y_j^{act}(t)$
d_2
d_1
0
k-1
k
d_2
t
$y_j^{act}(t)$
d_1
d_2
0
k-1
k
t

(a) When d_1 non-zero, and d_2 zero

(b) When d_1 zero, and d_2 non-zero

(c) When both zero

Figure 40: Then $(k):\ x_j(k)=\frac{1.5\cdot y_j^{act}(k-1)}{1.0}$, $(l):\ x_j(k)=\frac{1.5\cdot y_j^{act}(k)}{1.0}$, $(m):\ x_j(k)=\frac{1.0}{1.0}$

Computer Pseudo-Codes

These are computer pseudo-codes to implement the Cauchy-function variable $x_j(k) = (numerator/denominator)$.

```
IF($y_j^{act}(k) * y_j^{act}(k-1) > 0.0$)      : Case(I)
    IF($y_j^{act}(k) > y_j^{act}(k-1)$)
      $numerator = y_j^{act}(k);$ $denominator = y_j^{act}(k-1);$
    ELSE
      $numerator = y_j^{act}(k-1);$ $denominator = y_j^{act}(k);$
    END
ELSEIF($y_j^{act}(k) * y_j^{act}(k-1) < 0.0$)      : Case(II)
    IF($|y_j^{act}(k)| > |y_j^{act}(k-1)|$)
      $numerator = y_j^{act}(k);$ $denominator = y_j^{act}(k-1);$
    ELSE
      $numerator = y_j^{act}(k-1);$ $denominator = y_j^{act}(k);$
    END
ELSE      : Case(III)
    IF($(y_j^{act}(k) == 0) \& (y_j^{act}(k-1) \neq 0)$)
      $numerator = 1.5 \cdot y_j^{act}(k-1);$ $denominator = 1.0;$
    ELSEIF($(y_j^{act}(k) \neq 0) \& (y_j^{act}(k-1) == 0)$)
    ELSE
      $numerator = 1.0;$ $denominator = 1.0;$
    END
END
$\gamma_j(k) = \frac{C}{C+[numerator/denominator]^2};$
IF($\gamma_j(k) < 0.7$)
    $\gamma_j(k) = 0.7;$
END
```

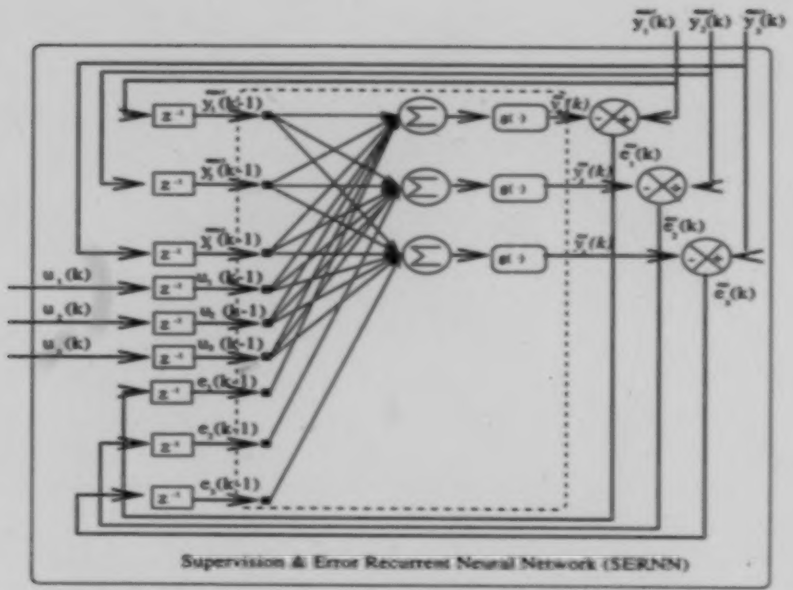

$\bar{y}_1(k)$
$\bar{y}_2(k)$
$\bar{y}_3(k)$
z^{-1}
$\bar{y}_1(k-1)$
$\bar{y}_2(k-1)$
$\bar{y}_3(k-1)$
$u_1(k)$
$u_2(k)$
$u_3(k)$
$u_1(k-1)$
$u_2(k-1)$
$u_3(k-1)$
$e_1(k-1)$
$e_2(k-1)$
$e_3(k-1)$
Σ
$g(\cdot)$
$\hat{y}_1(k)$
$\hat{y}_2(k)$
$\hat{y}_3(k)$
$\bar{e}_1(k)$
$\bar{e}_2(k)$
$\bar{e}_3(k)$
Supervision & Error Recurrent Neural Network (SERNN)

Figure 41: Single-Layer SERNN With Three Inputs/Outputs

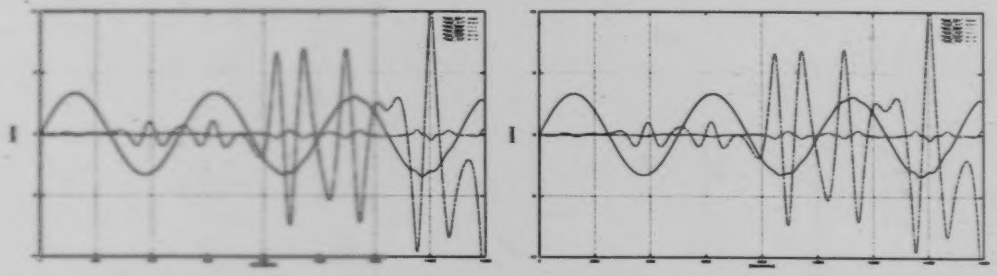
Performance Comparison Of the New Forgetting Factor

In order to compare the performance of the EW-CFF and the proposed EW-VFF developed in this thesis, the three-joint robot manipulator (PUMA 560) is used for the identification of the angular velocity in each joint. The SRLS algorithm with the CFF= 0.950 and the new VFF algorithm is applied to the neural model of a single layer SERNN[4] shown in Figure 41.

In computer simulation, the robot arm is supposed to pick up the payload at a random instance to reflect the dynamic uncertainty and the time-varying parameter estimates, and to track the variable trajectories. It is assumed that a payload of 2.3 Kg was picked up after 8 second (that is, 800^{th} iteration with sampling period of $0.01[sec]$) as shown in Figures 46, 47 48, and that the initial parameter values are reasonably chosen for the SRLS algorithm. Figure 42 shows the actual and model outputs of angular velocities for joint 1, 2, and 3 being identified by the neural network. Although left (a) and right (b) figures seem to look the same in Figure 42, the identification error shows the distinct differences for all joints in the comparison by Figures 46, 47, and 48. Figures 43, 44, and 45 represent the profiles of variable forgetting factors which are built on-line by the output measurement only.

---

[4] Supervisor and Error Recurrent Neural Network devised in the previous chapter



(a) With Conventional CFF= 0.950

(b) With Proposed VFF

Figure 42: Actual Outputs of Joint 1,2,3 Identified by Model Outputs

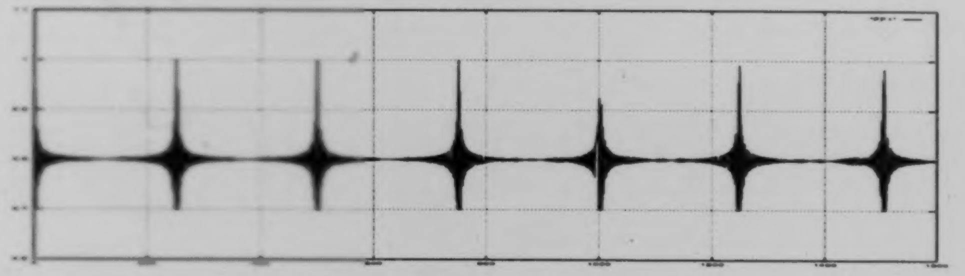


Figure 43: VFF Profile Used For Joint 1

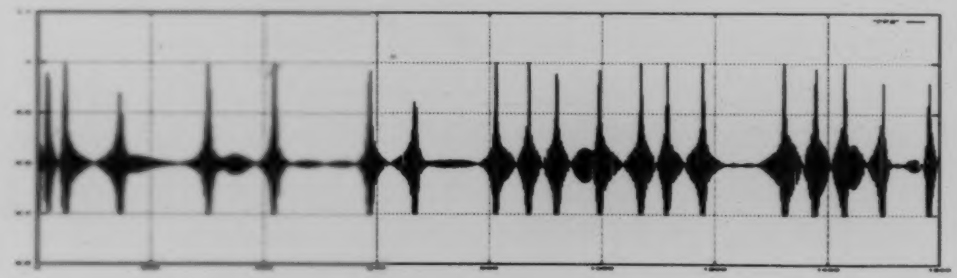


Figure 44: VFF Profile Used For Joint 2

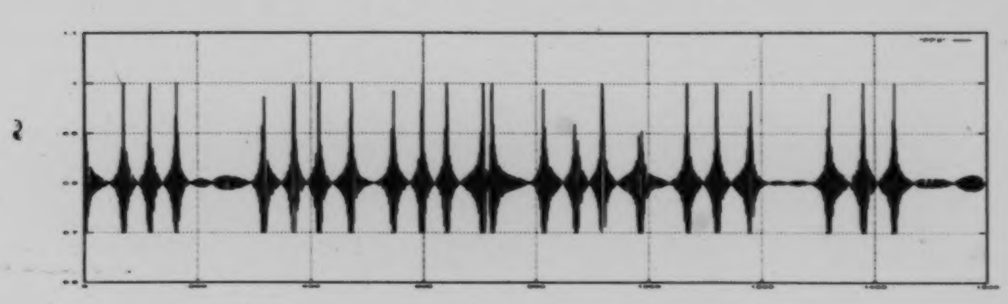
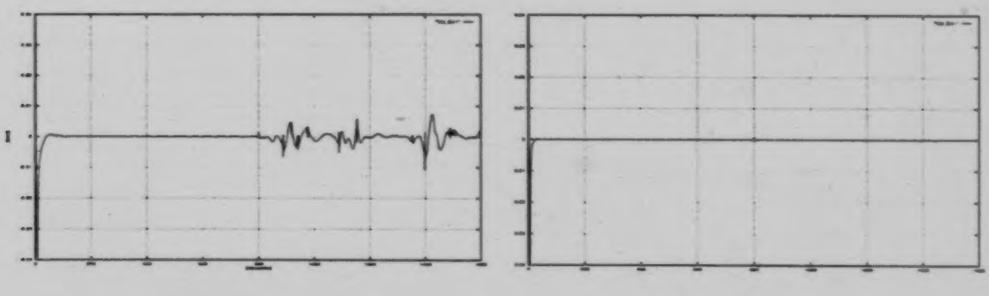


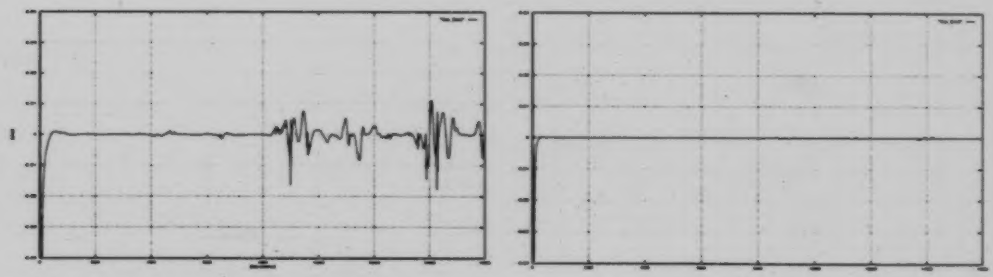
Figure 45: VFF Profile Used For Joint 3



(a) With Conventional CFF= 0.950

(b) With Proposed VFF

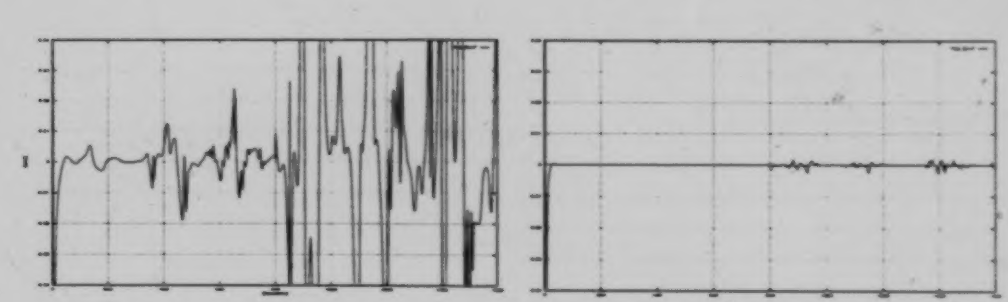
Figure 46: **Identification Error of Joint 1**



(a) With Conventional CFF= 0.950

(b) With Proposed VFF

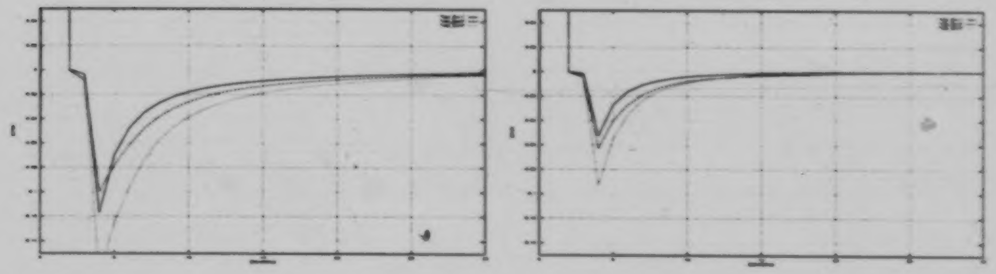
Figure 47: **Identification Error of Joint 2**



(a) With Conventional CFF= 0.950

(b) With Proposed VFF

Figure 48: **Identification Error of Joint 3**



(a) With Conventional CFF= 0.950

(b) With Proposed VFF

Figure 49: Initial Tracking Performances on Joint 1,2,3

As shown in Figures 46, 47, 48 and 49, the identification performance has clearly been improved in terms of the identification error and the initial tracking speed as shown in left (a) and right (b) figures. This research proposes for the first time the use of *Cauchy function* with argument of output measurement such as $\gamma_j(k) = c/c +$ [measurements]^2. Through this research, a *Cauchy constant* c is derived to yield a general constant value of $c \approx 5.0$ for different nonlinear plants even though it was chosen empirically.

5.4.7 Periodic Resetting of Covariance Matrix

In equation 5.23, the major problem with the forgetting factor is apparent that if the data vector $\mathbf{x}_j(k-1)$ does not contain much information, or in the extreme case of $\mathbf{x}_j(k-1) = 0$, then the covariance matrix $P_j(k)$ becomes:

$$P_j(k) = \frac{P_j(k-1)}{\gamma_j(k-1)} \tag{5.33}$$

and will grow exponentially. This causes the *estimator wind-up* or *burst phenomenon*. On the other hand, the adaptation gain in the *estimates update law* can converge to zero as time increases. It implies that the algorithm gradually loses its adaptability and will be turned off eventually as the algorithm runs. To avoid these phenomenon, this work reviews the following three techniques:

1. **Periodic resetting of the covariance matrix**. In this algorithm, the covariance matrix is generally reset to one of the old covariance matrices. The main problems are the choice of the resetting times and non-smooth nature of the resetting from the jumps. The *estimates update law* in the RLS algorithm should be robust enough to recover the lost information in the input vector by the periodic resetting. This methodology is mentioned in [8], [17], [2], and [12].

2. **Covariance matrix modification**. In this algorithm, some positive definite matrices are added to the *covariance matrix update law*. A certain perturbation term of the square error or the exogenous input is added to the update law as a driving force. These methods can achieve some of the desirable features, but unfortunately this is at the expense of other features such as a trade-off between the *alertness* and *burst phenomenon*. Some research on this concept is shown in [17], [3], and [12].

3. **Dynamic forgetting factor** is determined such that $\gamma_j(k) < 1$ when the situation of parameter change is detected, and $\gamma_k(k) = 1$ otherwise to prevent the covariance matrix from growing exponentially. Similar techniques are shown

in [8] and [13]. Under the incomplete persistent excitation, however, these can be prone to the *estimates wind-up*. [4] and [23] incorporate techniques to determine when parameter updating should be suspended so as to avoid wind-up when new information about parameters are not rich. The disadvantages of these clones are a delay of some steps in detecting a parameter change, tracking an abrupt variation, and the possibility of a *false alarm* or *failure* to detect a change.

One thus sees that the aforementioned techniques achieve some of the desirable features but unfortunately these are at the expense of other features. In this research, the following *periodic resetting of the covariance matrix* plus the *variable forgetting factor* developed in Section 5.4.6 is effective and avoids the *burst phenomenon* while keeping overall good performance of the MRLS algorithm. The periodic resetting of the covariance matrix update law is carried out every 35 iterations where 35 was chosen heuristically based on the fact that a certain robust SRLS estimation reaches the steady state after 25 to 35 iterations.

$$
\begin{aligned}
P_j(1) &= \frac{1}{\gamma_j(0)}\left[P_j(0) - \frac{P_j(0)\mathbf{x}_j(0)\mathbf{x}_j^T(0)P_j(0)}{\gamma_j(0) + \mathbf{x}_j^T(0)P_j(0)\mathbf{x}_j(0)}\right] \\
P_j(2) &= \frac{1}{\gamma_j(1)}\left[P_j(1) - \frac{P_j(1)\mathbf{x}_j(1)\mathbf{x}_j^T(1)P_j(1)}{\gamma_j(1) + \mathbf{x}_j^T(1)P_j(1)\mathbf{x}_j(1)}\right] \\
&\qquad \vdots \qquad\qquad \vdots \\
\rightarrow P_j(35) &= \frac{1}{\gamma_j(34)}\left[P_j(34) - \frac{P_j(34)\mathbf{x}_j(34)\mathbf{x}_j^T(34)P_j(34)}{\gamma_j(34) + \mathbf{x}_j^T(34)P_j(34)\mathbf{x}_j(34)}\right] \\
&\qquad \vdots \qquad\qquad \vdots \\
\rightarrow P_j(70) &= \frac{1}{\gamma_j(69)}\left[P_j(35) - \frac{P_j(35)\mathbf{x}_j(69)\mathbf{x}_j^T(69)P_j(35)}{\gamma_j(69) + \mathbf{x}_j^T(69)P_j(35)\mathbf{x}_j(69)}\right] \\
P_j(71) &= \frac{1}{\gamma_j(70)}\left[P_j(70) - \frac{P_j(70)\mathbf{x}_j(70)\mathbf{x}_j^T(70)P_j(70)}{\gamma_j(70) + \mathbf{x}_j^T(70)P_j(70)\mathbf{x}_j(70)}\right] \\
&\qquad \vdots \qquad\qquad \vdots
\end{aligned}
$$

$$\rightarrow P_j(105) = \frac{1}{\gamma_j(104)}\left[P_j(35) - \frac{P_j(35)\mathbf{x}_j(104)\mathbf{x}_j^T(104)P_j(35)}{\gamma_j(104) + \mathbf{x}_j^T(104)P_j(35)\mathbf{x}_j(104)}\right]$$

$$P_j(106) = \frac{1}{\gamma_j(105)}\left[P_j(105) - \frac{P_j(105)\mathbf{x}_j(105)\mathbf{x}_j^T(105)P_j(105)}{\gamma_j(105) + \mathbf{x}_j^T(105)P_j(105)\mathbf{x}_j(105)}\right]$$

$$\vdots \qquad\qquad \vdots$$

The following computer simulation clearly shows the performance improvement by means of both the **periodic resetting of covariance matrix** and the developed **variable forgetting factor**. For comparison purposes on the **wind-up problem** of the covariance matrix, the following two methods are tested on a plant as shown in Figure 50 to identify the system output.

- Conventional Methods:
    1. Test plant: SISO shown in Figure 50 with **poor persistent excitation**.
    2. Neural model: Two-layer conventional RNN (bias=+1, self-recurrent).
    3. Linearization: Linear observation by inverse of the activation function in Section 5.5.1.
    4. Algorithm for training: Estimates update law in the SRLS algorithm.
    5. Hidden layer teaching: Clear derivation of an optimal teaching signal shown in Section 5.5.2 (Novel).
    6. Forgetting factor: Exponential-weight constant f.f (EW-CFF = 0.98).
    7. **Covariance matrix update: Covariance update law in the SRLS algorithm plus EW-CFF**.
- Proposed Methods:
    1. Test plant: SISO shown in Figure 50 with **poor persistent excitation**.
    2. Neural model: Two-layer SERNN (Novel).
    3. Linearization: Linear observation by inverse of the activation function in Section 5.5.1.

4. Algorithm for training: Estimates update law in MRLS algorithm (Novel).
5. Hidden layer teaching: Clear derivation of an optimal teaching signal shown in Section 5.5.2 (Novel).
6. Forgetting factor: Exponential-weight variable f.f (EW-VFF: Novel) .
7. **Covariance matrix update: Periodic resetting of covariance update law** (same as SRLS) **in the MRLS algorithm plus EW-VFF**.

The **clear and present performance improvement** is shown in Figures 51 to 57. Figure 51 (a) shows the *exponential growth* of covariance matrix by the conventional method due to the prospective *poor persistent excitation*, while 51 (b) represents the proposed algorithm properly *alive* in the output layer. Thereby, the overall *identification wind-up* is clearly shown in Figures 53(a), 54(a) by use of the conventional methodologies. But both methods show the *alertness* of the covariance matrix in the hidden layer as shown in Figure 52. Besides the problem of the covariance divergence, the proposed novel algorithm may improve the initial training (estimation) speed represented by Figure 54. Even though the conventional method can adopt the same periodic resetting of covariance matrix in conjunction with SRLS algorithm, it frequently goes to the same *burst phenomenon*. This fact implies that the novel MRLS algorithm is more robust than the SRLS algorithm with respect to the lost information.

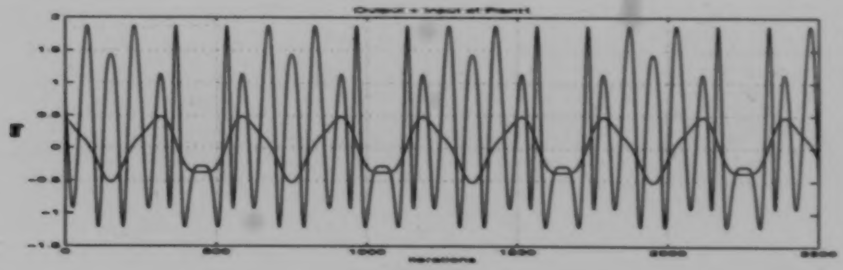
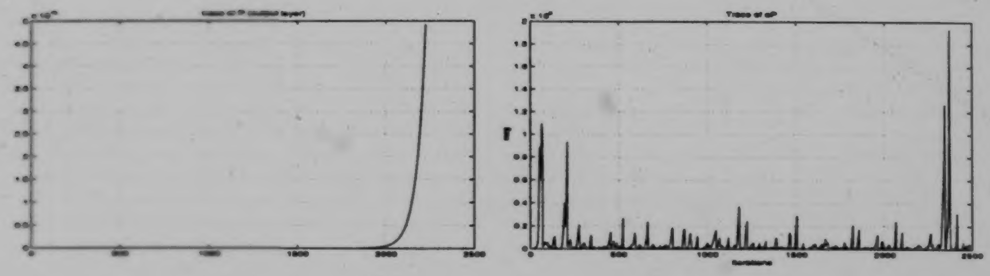


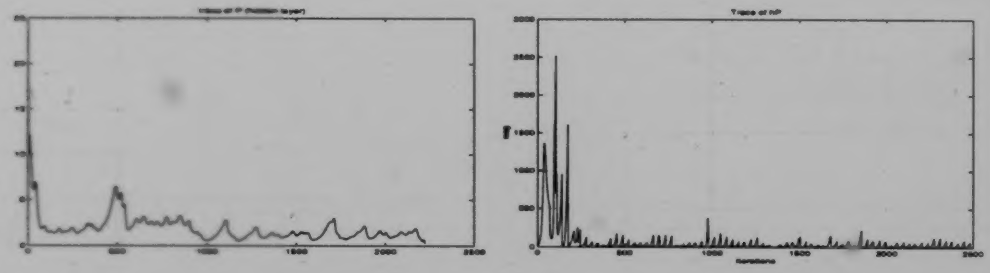
Figure 50: Input (short) and output (tall) for a test plant



(a) by conventional updating with CFF (**Divergent**)

(b) by proposed periodic resetting with VFF (**Alert**)

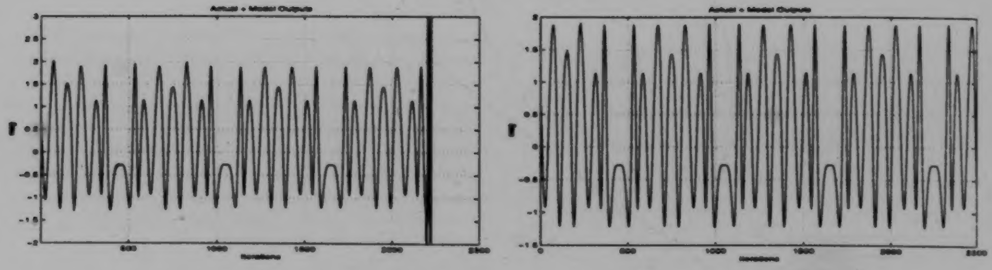
Figure 51: Trace of covariance matrix in output layer



(a) by conventional updating with CFF

(b) by proposed periodic resetting with VFF (**Alert**)

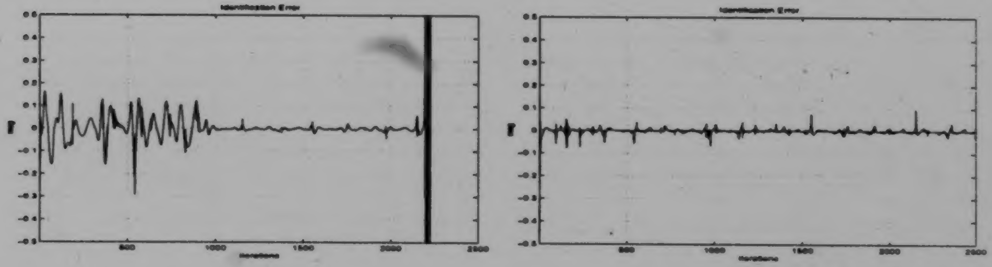
Figure 52: Trace of covariance matrix in hidden layer



(a) by conventional method (**Wind-up**)

(b) by proposed method

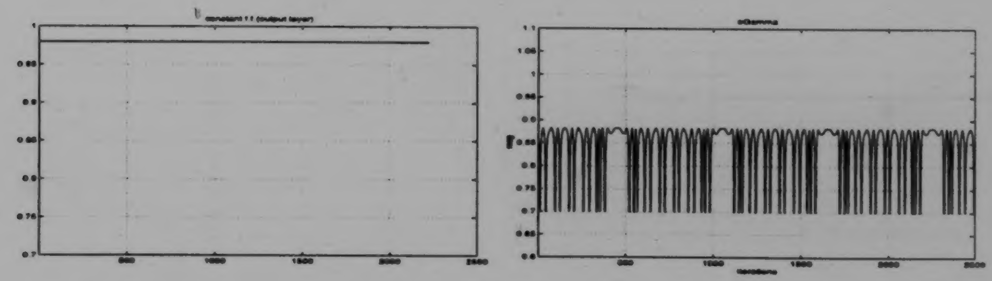
Figure 53: Plant output identified by conventional and proposed methods



(a) by conventional method (**Burst**)

(b) by proposed method

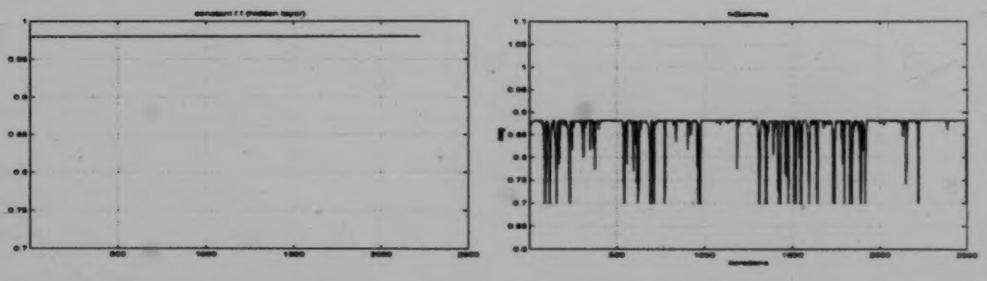
Figure 54: Output identification error



(a) by conventional CFF= 0.98

(b) by proposed VFF

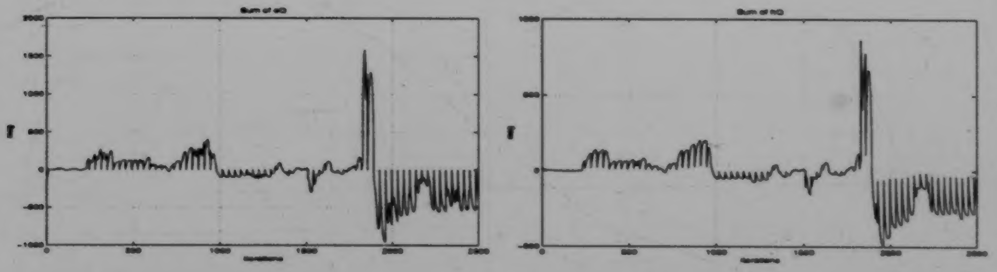
Figure 55: Forgetting factor profiles in output layer



(a) by conventional CFF= 0.98

(b) by proposed VFF

Figure 56: Forgetting factor profiles in hidden layer



(a) Trace of $Q(k)$ in output layer

(b) Trace of $Q(k)$ in hidden layer

Figure 57: Trace of input vector update of $Q(k)$ for MRLS algorithm

5.5 Application of MRLS to Neural Training

5.5.1 Linearization of Neural Model

The RLS algorithm clone can only be applied to a linear parametric form to estimate the unknown parameters in the model. One way to apply the RLS algorithms to the neural networks is to linearize the nonlinear neural model or to observe the linear part by dividing a neuron into the linear and the nonlinear blocks. This section introduces two methods called the ***Taylor series*** expansion derived effectively in this thesis for the recursive estimation algorithms, and the ***linear observation*** by the inverse activation function techniques. Both methods are suitable for practical neural training.

Linearization by Taylor Series

A nonlinear neuron characterized by the nonlinear *activation function* is linearized by the use of the *Taylor series* such that the activation function $g(\cdot)$ is expanded with respect to the previous weight vector $\hat{\mathbf{w}}(k-1)$, which is known at the present sampling instance k. The following processes show the detailed *Taylor expansion* on the neuron output.

Let $y_j^{nn}(k)$ be the jth neuron output, which is differentiated with respect to the known weight estimate of $\hat{\mathbf{w}}(k-1)$ as shown in equation 5.34.

(5.34)

$$\begin{aligned}
y_j^{nn}(k) &= g[v] \\
&= g\Big[\mathbf{x}_j^T(k-1)\cdot\mathbf{w}_j(k)\Big] \\
&= g\left[\mathbf{x}_j^T(k-1)\cdot\hat{\mathbf{w}}_j(k-1)\right] + \left[\frac{\partial g(v)}{\partial \mathbf{w}_j(k)}\right]\cdot\left[\mathbf{w}_j(k)-\hat{\mathbf{w}}_j(k-1)\right] + e_j^{hot}(k) \\
&= g\left[\mathbf{x}_j^T(k-1)\cdot\hat{\mathbf{w}}_j(k-1)\right] + \left[\frac{dg(v)}{dv}\right]\mathbf{x}_j^T(k-1)\left[\mathbf{w}_j(k)-\hat{\mathbf{w}}_j(k-1)\right] + e_j^{hot}(k) \\
&= g\left[\mathbf{x}_j^T(k-1)\cdot\hat{\mathbf{w}}_j(k-1)\right] + \mathbf{q}_j^T(k-1)\cdot\left[\mathbf{w}_j(k)-\hat{\mathbf{w}}_j(k-1)\right] + e_j^{hot}(k) \\
&= \mathbf{q}_j^T(k-1)\cdot\mathbf{w}_j(k) + g\left[\mathbf{x}_j^T(k-1)\hat{\mathbf{w}}_j(k-1)\right] - \mathbf{q}_j^T(k-1)\hat{\mathbf{w}}_j(k-1) + e_j^{hot}(k)
\end{aligned}$$

where $e_j^{hot}(k)$ is the residual error of the *higher-order-terms* occurring in the process of Taylor series expansion, and

$$\mathbf{q}_j^T(k-1) \triangleq g'(v)|_{\mathrm{at}\,\hat{\mathbf{w}}_j(k-1)}\cdot\mathbf{x}_j^T(k-1)$$

For a hyperbolic tangent activation function:

$$\begin{aligned}
g(v) &= b\cdot\tanh(\frac{a}{2}\cdot v), \qquad b:\text{ Saturation Limit and } a:\text{ Slope,} \\
&= b\cdot\left[\frac{1-e^{-av}}{1+e^{-av}}\right] \\
g'(v) &= \frac{2ab\cdot e^{-av}}{(1+e^{-av})^2}
\end{aligned}$$

$$\therefore\ \mathbf{q}_j^T(k-1) \triangleq 2 \cdot a \cdot b \cdot \left[\frac{e^{-a \cdot \mathbf{x}_j^T(k-1) \cdot \hat{\mathbf{w}}_j(k-1)}}{\left(1 + e^{-a \cdot \mathbf{x}_j^T(k-1) \cdot \hat{\mathbf{w}}_j(k-1)}\right)^2} \right] \cdot \mathbf{x}_j^T(k-1)$$

Then, the neural training is described by the process of minimizing the modeling error $e_j^{mo}(k)$ between the teaching signal (actual measurement) $y_j^{act}(k)$ and the neural network output $y_j^{nn}(k)$ shown in equation 5.35.

$$y_j^{act}(k) = y_j^{nn}(k) + e_j^{mo}(k) \tag{5.35}$$

By combining equations 5.35 and 5.34, and rearranging the **linearized model** on a *jth* neuron output is written as.

$$\therefore\ \boxed{z_j(k) = \mathbf{q}_j^T(k-1) \cdot \mathbf{w}_j(k) + e_j^{hm}(k)} \tag{5.36}$$

where $z_j(k)$ and $e_j^{hm}(k)$ are defined by equations 5.37 and 5.38 respectively.

$$z_j(k) \triangleq y_j^{act}(k) - g[\mathbf{x}_j^T(k-1) \cdot \hat{\mathbf{w}}_j(k-1)] + \mathbf{q}_j^T(k-1) \cdot \hat{\mathbf{w}}_j(k-1) \tag{5.37}$$

$$e_j^{hm}(k) \triangleq e_j^{hot}(k) + e_j^{mo}(k) \tag{5.38}$$

Now $z_j(k)$ and $\mathbf{q}_j^T(k-1)$ are computable signals at the current sampling instance k. Based on equation 5.36, the neural training process is interpreted such that the filtered signals $z_j(k)$ and $\mathbf{q}_j^T(k-1)$ are used for a *new teaching signal* and a *new input vector* respectively to estimate the unknown synaptic weight $\mathbf{w}_j(k)$. By making use of the prerequisites in terms of a linear parametric model, a teaching signal and an input vector, the RLS algorithm can be carried out to minimize both the linearizing error $e_j^{hot}(k)$ and the modeling error $e_j^{mo}(k)$. This linearizing method turns out to be very accurate for time-varying systems with reasonable sampling frequency compared to the similar way shown in [7].

Linear Observation By Inverse Activation Function

Referring to Figure 58, [19] suggests that a nonlinear neural network problem could be partitioned into linear and nonlinear portions. This means that if all inputs to summation and summation outputs were specified, the problem would be reduced to a linear problem, i.e., a system of linear equations that relates the summation outputs to the weight vector and node inputs. In other words, the training process uses the supervising signal v_j^{act} and the neuron output $v_j^{nn}(k)$ in the linear part instead of the $y_j^{act}(k)$ and $y_j^{nn}(k)$ in the nonlinear part as shown in Figure 58, provided that the activation function $g(\cdot)$ is invertible. This *linear observation* can be considered as one linearization method for a nonlinear neuron model by taking the inverse activation function of supervising signal. The main advantage of this methodology is its simplicity in implementation in the presence of an activation function inverse.

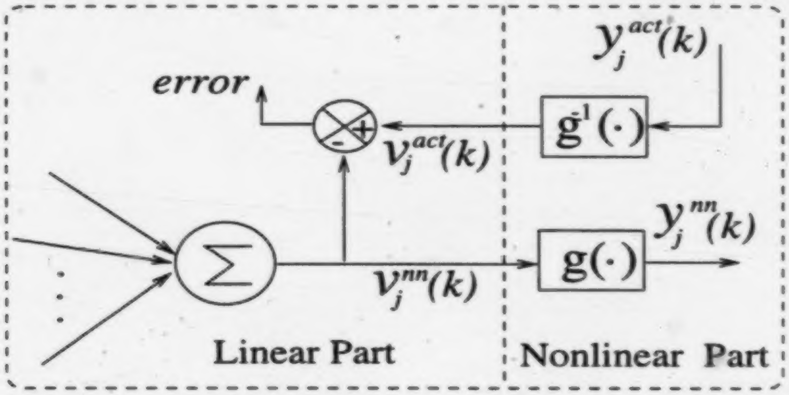


Figure 58: Linear Observation of A Neuron

The application of the linear observation is shown in [19] and [10] to train the multilayer neural networks. They used the *Kalman filtering* method for better convergence rate and less sensitivity to the initial weight values. However, they use the backpropagation algorithm to train the hidden layer, which still gives rise to similar shortcomings from the use of backpropagation clone. The following section proposes a **new teaching signal** for the **hidden-layer training** based on the optimal method. The new hidden-layer teaching method is expected to be free from most of BP drawbacks.

5.5.2 Derivation Of Explicit Supervisory Signal For Hidden Layers

A network can have several layers. A layer that produces the network output is called an *output layer.* All other layers are called *hidden layers.* Even though multiple layer networks are quite powerful compared to single layer ones, their training method is generally more difficult due to the problem of hidden layer training. The main obstacle in training the multi-layer ANNs has been the *absence of a teaching (desired) output for the hidden layers.* This has been treated by the *error backpropagation* algorithm or *generalized delta rule* developed by [16]. Although the BP algorithm has worked successfully for a wide variety of applications, general BP learning algorithms have several limitations. The long and unpredictable training process is the most troublesome, for example the rate of convergence is seriously affected by the initial weights and the learning rate of the parameters. In general, increasing the learning step size can speed up the convergence rate of the learning process, but it may also lead to divergence, paralysis, or continuous instability. Many research activities focus on how to increase the learning rate. The learning rate can be especially important in real-time applications, where the training set is not known in advance, may be time-varying and non-repeatable. In this research, the derivation of a clear supervisory signal for the hidden layer is tackled to overcome the drawbacks of error BP algorithms.

Referring to Figure 59, the mathematical form of the teaching signal for the hidden layer is derived to minimize the residual error in the output layer with respect to the hidden layer output vector ${}^{(h)}Y^{nn}(k-1)$. The cost function for the *linear least-squares* (L_2-norm) problem (which is a special case of the nonlinear least-squares problem) is formulated as follows:

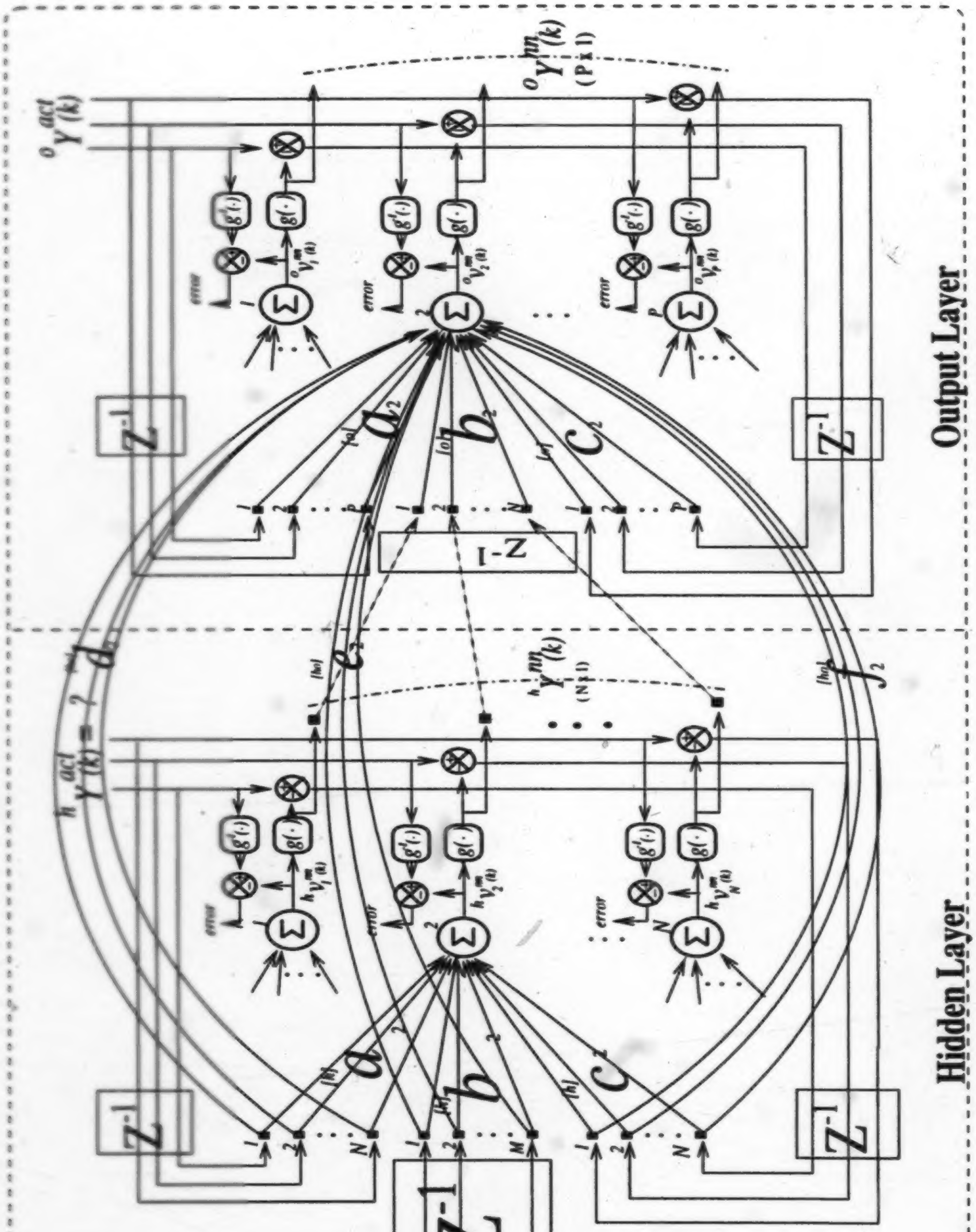

Output Layer
Hidden Layer
$^{o}Y^{act}(k)$
$^{o}Y^{nn}(k)$
(P x 1)
$^{h}Y^{act}(k) = ?$
$^{h}Y^{nn}(k)$
(N x 1)
Z^{-1}
error
$g(\cdot)$
$g'(\cdot)$
Σ
a_2
b_2
c_2
d_2
e_2
f_2

Figure 59: Schematic Diagram of Multi-Layer SERNN

(5.39)

$$
\begin{aligned}
J[{}^{(h)}Y^{nn}(k-1)] &= \frac{1}{2}\left\| g^{-1}[{}^{(o)}Y^{act}(k)] - g^{-1}[{}^{(o)}Y^{nn}(k)] \right\|_2^2 \\
&= \frac{1}{2}\left\| {}^{(o)}V^{act}(k) - {}^{(o)}V^{nn}(k) \right\|_2^2 \\
&= \frac{1}{2}\Big\| {}^{(o)}V^{act}(k) - \Big\{ {}^{(o)}\hat{A}^T \cdot {}^{(o)}Y^{act}(k-1) + {}^{(o)}\hat{B}^T \cdot {}^{(h)}Y^{nn}(k-1) + \\
&\quad {}^{(o)}\hat{C}^T \cdot {}^{(o)}\mathcal{E}(k-1) + {}^{(ho)}\hat{D}^T \cdot {}^{(h)}Y^{act}(k-1) + {}^{(ho)}\hat{E}^T \cdot {}^{(h)}U(k-1) + \\
&\quad {}^{(ho)}\hat{F}^T \cdot {}^{(h)}\mathcal{E}(k-1) \Big\} \Big\|_2^2
\end{aligned}
$$

where $g^{-1}[\cdot]$ is the inversion of the activation function; ${}^{(o)}Y^{act}(k) \in \Re^{P\times1}$ and ${}^{(o)}Y^{nn}(k) \in \Re^{P\times1}$ are the actual teaching signal and the neural output in the output layer respectively, ${}^{(o)}V^{act}(k) \in \Re^{P\times1}$ and ${}^{(o)}V^{nn}(k) \in \Re^{P\times1}$ are the linearized signals in the output layer, ${}^{(o)}\mathcal{E}(k) \in \Re^{P\times1}$ is the modeling error between ${}^{(o)}Y^{act}(k)$ and ${}^{(o)}Y^{nn}(k)$ in the output layer, ${}^{(h)}\mathcal{E}(k) \in \Re^{P\times1}$ is the modeling error between ${}^{(h)}Y^{act}(k)$ and ${}^{(h)}Y^{nn}(k)$ of the hidden layer, ${}^{(h)}U(k) \in \Re^{M\times1}$ is the (external) input vector in the hidden layer, superscript of ${}^{(ho)}$ stands for *inter-layer* weights between hidden and output layers in the forward direction, P, N, and M are the numbers of output nodes in the output layer, output nodes in the hidden layer, and external inputs in the hidden layer respectively, and ${}^{(o)}\hat{A}$, ${}^{(o)}\hat{B}$, ${}^{(o)}\hat{C}$, ${}^{(ho)}\hat{D}$, ${}^{(ho)}\hat{E}$, ${}^{(ho)}\hat{F}$, are the weight matrices, which are known from the first training in the output layer. They are shown below in conjunction with Figure 59:

$$
{}^{(o)}\hat{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1P} \\ a_{21} & a_{22} & \cdots & a_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ a_{P1} & a_{P2} & \cdots & a_{PP} \end{bmatrix}_{P\times P}^{T} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_P \end{bmatrix} \tag{5.40}
$$

$$\mathbf{a}_1 = \begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{1P} \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} a_{21} \\ a_{22} \\ \vdots \\ a_{2P} \end{bmatrix}, \quad \cdots \quad , \mathbf{a}_P = \begin{bmatrix} a_{P1} \\ a_{P2} \\ \vdots \\ a_{PP} \end{bmatrix}$$

(5.41)
$$^{(o)}\hat{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1N} \\ b_{21} & b_{22} & \cdots & b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{P1} & b_{P2} & \cdots & b_{PN} \end{bmatrix}^T_{N\times P} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_P \end{bmatrix}$$

$$\mathbf{b}_1 = \begin{bmatrix} b_{11} \\ b_{12} \\ \vdots \\ b_{1N} \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} b_{21} \\ b_{22} \\ \vdots \\ b_{2N} \end{bmatrix}, \quad \cdots \quad , \mathbf{b}_P = \begin{bmatrix} b_{P1} \\ b_{P2} \\ \vdots \\ b_{PN} \end{bmatrix}$$

(5.42)
$$^{(o)}\hat{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1P} \\ c_{21} & c_{22} & \cdots & c_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ c_{P1} & c_{P2} & \cdots & c_{PP} \end{bmatrix}^T_{P\times P} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_P \end{bmatrix}$$

$$\mathbf{c}_1 = \begin{bmatrix} c_{11} \\ c_{12} \\ \vdots \\ c_{1P} \end{bmatrix}, \quad \mathbf{c}_2 = \begin{bmatrix} c_{21} \\ c_{22} \\ \vdots \\ c_{2P} \end{bmatrix}, \quad \cdots \quad , \mathbf{c}_P = \begin{bmatrix} c_{P1} \\ c_{P2} \\ \vdots \\ c_{PP} \end{bmatrix}$$

$$(5.43)\qquad {}^{(ho)}\hat{D} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N} \\ d_{21} & d_{22} & \cdots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{P1} & d_{P2} & \cdots & d_{PN} \end{bmatrix}^{T}_{N\times P} = \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_P \end{bmatrix}$$

$$\mathbf{d}_1 = \begin{bmatrix} d_{11} \\ d_{12} \\ \vdots \\ d_{1N} \end{bmatrix}, \quad \mathbf{d}_2 = \begin{bmatrix} d_{21} \\ d_{22} \\ \vdots \\ d_{2N} \end{bmatrix}, \quad \cdots \quad , \mathbf{d}_P = \begin{bmatrix} d_{P1} \\ d_{P2} \\ \vdots \\ d_{PN} \end{bmatrix}$$

$$(5.44)\qquad {}^{(ho)}\hat{E} = \begin{bmatrix} e_{11} & e_{12} & \cdots & e_{1M} \\ e_{21} & e_{22} & \cdots & e_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ e_{P1} & e_{P2} & \cdots & e_{PM} \end{bmatrix}^{T}_{M\times P} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_P \end{bmatrix}$$

$$\mathbf{e}_1 = \begin{bmatrix} e_{11} \\ e_{12} \\ \vdots \\ e_{1M} \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} e_{21} \\ e_{22} \\ \vdots \\ e_{2M} \end{bmatrix}, \quad \cdots \quad , \mathbf{e}_P = \begin{bmatrix} e_{P1} \\ e_{P2} \\ \vdots \\ e_{PM} \end{bmatrix}$$

$$
{}^{(ho)}\hat{F} = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ f_{21} & f_{22} & \cdots & f_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ f_{P1} & f_{P2} & \cdots & f_{PN} \end{bmatrix}^{T}_{N\times P} = \begin{bmatrix} \mathbf{f}_1 & \mathbf{f}_2 & \cdots & \mathbf{f}_P \end{bmatrix} \tag{5.45}
$$

$$
\mathbf{f}_1 = \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{1N} \end{bmatrix}, \quad \mathbf{f}_2 = \begin{bmatrix} f_{21} \\ f_{22} \\ \vdots \\ f_{2N} \end{bmatrix}, \quad \cdots \quad , \mathbf{f}_P = \begin{bmatrix} f_{P1} \\ f_{P2} \\ \vdots \\ f_{PN} \end{bmatrix}
$$

Now, the *linear least-squares* (L_2 norm) technique is applied to the equation 5.39 to derive the optimal teaching signal of ${}^{(h)}Y^{act}(k)^*$ by setting up equation 5.46.

$$\frac{\partial J}{\partial\, {}^{(h)}Y^{nn}(k-1)} = 0 \tag{5.46}$$

The solution of equation 5.46 can be grouped into three categories depending on the size of P and N, that is, the relation of the number of equations and the number of variables to be solved.

1. **Case I** : The number of the neurons in the output layer is the same as that in the hidden layer.
2. **Case II** : The number of the neurons in the output layer is larger than that in the hidden layer.
3. **Case III** : The number of the neurons in the output layer is smaller than that in the hidden layer.

The solutions are shown as the following mathematical forms:

1. $\underline{\textbf{Case I} : rank(\mathbf{B}) = N = P} \Longrightarrow$ A unique solution exists:

   (5.47)
   $$\begin{aligned}{}^{(h)}Y^{nn}(k-1)^* = \left[{}^{(o)}\hat{B}^T\right]^{-1} \cdot \Big[&{}^{(o)}V^{act}(k) - {}^{(o)}\hat{A}^T \,.\, {}^{(o)}Y^{act}(k-1) - {}^{(o)}\hat{C}^T \,.\, {}^{(o)}\mathcal{E}(k-1) - \\ &{}^{(ho)}\hat{D}^T \,.\, {}^{(h)}Y^{act}(k-1) - {}^{(ho)}\hat{E}^T \,.\, {}^{(h)}U(k-1) - {}^{(ho)}\hat{F}^T \,.\, {}^{(h)}\mathcal{E}(k-1)\Big]\end{aligned}$$

   with $J[{}^{(h)}Y^{nn}(k-1)^*] = 0$,

2. $\underline{\textbf{Case II} : rank(\mathbf{B}) = N < P} \Longrightarrow$ An exact solution does not exist, but the least-squares error solution is:

   (5.48)
   $$\begin{aligned}{}^{(h)}Y^{nn}(k-1)^* = \left[{}^{(o)}\hat{B}\,{}^{(o)}\hat{B}^T\right]^{-1} \cdot {}^{(o)}\hat{B}\Big[&{}^{(o)}V^{act}(k) - {}^{(o)}\hat{A}^T \,.\, {}^{(o)}Y^{act}(k-1) - {}^{(o)}\hat{C}^T \,. \\ &{}^{(o)}\mathcal{E}(k-1) - {}^{(ho)}\hat{D}^T \,.\, {}^{(h)}Y^{act}(k-1) - {}^{(ho)}\hat{E}^T \,.\, {}^{(h)}U(k-1) - \\ &{}^{(ho)}\hat{F}^T \,.\, {}^{(h)}\mathcal{E}(k-1)\Big]\end{aligned}$$

where $[{}^{(o)}\hat{B}{}^{(o)}\hat{B}^T]^{-1} \cdot {}^{(o)}\hat{B}$ is called the *Moore-Penrose* pseudo-inverse [6] with $J[{}^{(h)}Y^{nn}(k-1)^{*}] \geqq 0$,

3. **Case III** : $rank(\mathbf{B}) = N > P \Longrightarrow$ The solution is not unique and the minimum L_2-norm solution is:

   (5.49)

$$
\begin{aligned}
{}^{(h)}Y^{nn}(k-1)^{*} = {}^{(o)}\hat{B}\left[{}^{(o)}\hat{B}^{T(o)}\hat{B}\right]^{-1} \cdot \Big[&{}^{(o)}V^{act}(k) - {}^{(o)}\hat{A}^T \,.\, {}^{(o)}Y^{act}(k-1) - {}^{(o)}\hat{C}^T \,. \\
&{}^{(o)}\mathcal{E}(k-1) - {}^{(ho)}\hat{D}^T \,.\, {}^{(h)}Y^{act}(k-1) - {}^{(ho)}\hat{E}^T \,.\, {}^{(h)}U(k-1) - \\
&{}^{(ho)}\hat{F}^T \,.\, {}^{(h)}\mathcal{E}(k-1)\Big]
\end{aligned}
$$

   with $J[{}^{(h)}Y^{nn}(k-1)^{*}] = 0$,

then, the optimal teaching signal of ${}^{(h)}Y^{act}(k)^{*}$ for the hidden layer becomes:

(5.50)
$$\boxed{{}^{(h)}Y^{act}(k)^{*} = {}^{(h)}Y^{nn}(k-1)^{*}}$$

It is noted that ${}^{(h)}Y^{nn}(k-1)$ (its own previous output of the hidden layer) and ${}^{(h)}Y^{nn}(k-1)^{*}$ (desirable previous output of the hidden layer minimizing the present training error in the output layer) in equations 48 and 5.50 are different signals. From equations 5.47, 5.48, 5.49 and 5.50, the **hidden layer's clear teaching signal** has been derived for on-line training based on the minimization of the L_2-norm. The derivation of the hidden layer's teaching signal has a very important meaning for the training of ***multi-layer ANNs*** (ML-ANNs). So far, the error BP algorithm has been used for ML-ANNs training even though there exists lots of known shortcomings in BP because of the ***absence of the reasonable hidden layer's teaching signal.*** In this author's opinion, the hidden layer's teaching method, except for using the error BP clone, has not been researched.

5.6 Remarks On MRLS Algorithm For ANNs Training

In order to improve the performance of the error BP algorithms, the use of the SRLS algorithm is proposed for the training of the ML-ANNs. However, since the SRLS algorithm still has undesirable features, such as the numerical wind-up, the MRLS algorithm was derived with the improved characteristics on the variable forgetting factor, the robust resetting of the covariance matrix along with EW-VFF and the hidden layer teaching by the clear supervision signal in the hidden layer. The combination of the novel MRLS algorithm in this chapter and the new SERNN architecture developed in the previous chapter has desired features such as fast initial learning speed, robustness, and accuracy to minimize the mismatch (called *uncertainty*) between the behavior of the real system and that expected from the ANNs model. These improved performances will be shown in the next Chapter through the computer simulation for **nonlinear system identification** which has been known as a crucial part in the adaptive processing area.

Bibliography

[1] Peter Anderson. Adaptive forgetting in recursive identification through multiple models. *Int. J. Control*, vol.42(no.5):pp.1175–1193, 1985.

[2] K.J. Åström and B. Wittenmark. *Adaptive Control.* Addison-Wesley, 1989.

[3] S. Bittanti, P. Bolzern, and M. Campi. Recursive least-squares identification algorithms with incomplete excitation: convergence analysis and application to adaptive control. *Trans. on Automatic Control*, vol.25(no.12):pp.1371–1373, Dec. 1990.

[4] M.J. Chen and J.P. Norton. Estimation technique for tracking rapid parameter changes. *Int. J. Control*, vol.45(no.4):pp.1387–1398, 1987.

[5] R. Isermann (Ed.). *System Identification.* Pergamon Press, Darmstadt, Oxford, 1981. Tutorials presented at the 5th IFAC symposium on identification and system parameter estimation.

[6] G.H Golub and C.F. Van Loan. *Matric Computation.* Johns Hopkins University Press, Baltimore, MD, 1989.

[7] Simon Haykin. *Neural Networks : A Comprehensive Foundation.* Macmillan College Publishing Company Inc., New York, 1994.

[8] K.J. Hunt. A survey of resursive identification algorithms. *Trans. Inst. MC*, vol.8(no.5):pp.273–278, Oct–Dec 1986.

[9] A.J. Koivo. *Fundamentals for Control of Robotic Manipulators.* John Wiley & Sons, Inc., 1989.

[10] K.-N. Lou and R.A. Perez. A new system identification technique using kalman filtering and multilayer neural networks. *Artificial Intelligence in Engineering*, vol.10:pp.1–8, 1996.

[11] Michael McInerney and Atam P. Dhawan. Use of genetic algorithms with backpropagation in training of feedforward neural networks. *IEEE Int. Conf. on Neural Networks*, vol.1, 1993.

[12] D.J. Park and B.E. Jun. Selfperturbing recursive least squares algorithm with fast tracking capability. *Electronics Letters*, vol.28(no.6):pp.558–559, 12th March 1992.

[13] D.J. Park, B.E. Jun, and J.H. Kim. Fast tracking rls algorithm using novel variable forgetting factor with unity zone. *Electronics Letters*, vol.27(no.23):pp.2150–2151, 7th November 1991.

[14] B.A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, vol.1(no.2):pp.263–269, 1989.

[15] F. Pineda. Time dependent adaptive neural networks. In D.S. Touretzky, editor, *Advances in neural information processing systems*, volume 2, pages 710–718, San Mateo, CA, 1990. Morgan Kaufmann.

[16] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group. *Parallel distributed processing (PDP): Exploration in the microstructure of cognition*, volume vol.1. MIT Press, Cambridge, MA, 1986.

[17] M.E. Salgado, G.C Goodwin, and R.H. Middleton. Modified least squares algorithm incorporating exponential resetting anf forgetting. *Int. J. Control*, vol.47(no.2):pp.477–491, 1988.

[18] M. Sato. A real time learning algorithm for recurrent analog neural networks. *Biological Cybernetics*, vol.62(no.3):pp.237–242, 1990.

[19] Robert S. Scalero and Nazif Tepedelenlioglu. A fast new algorithm for training feedforward neural networks. *IEEE Trans. on Signal-processing*, vol.40(no.1):pp.202–210, 1992.

[20] S.L. Shah. Rls estimation schemes for adaptive control. *IEE Colloquium on Advances in Adaptive Control*, April 1986. London.

[21] S. Singhal and L. Wu. Training feed-forward networks with the extended kalman algorithm. *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages 1187–1190, May 1989.

[22] N. Toomarian and J. Barhen. Adjoint operators and nonadiabatic algorithms in neural networks. *Applied Mathematics Letters*, vol.4(no.2):pp.69–73, 1991.

[23] H. Wang and S. Daley. An algorithm for tracking rapid parameter changes in unknown mimo systems with nonlinear model uncertainties. *International Conference on Control'91*, vol.1:pp.551–516, 25–28, March 1991.

[24] P.D. Wasserman. *Experiments in translating chinese characters using backpropagation*. Van Nostrand Reinhold, New York, 1988.

[25] B. Widrow and S.D. Sterns. *Adaptive Signal Processing*. Prentice-Hall, Inc., Englewood Cliffs, 1985.

[26] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, vol.1:pp.270–280, 1989.

Chapter 6

Simulations Of Neural Nonlinear System Identification

6.1 Objectives Of Simulation

The previous chapters have discussed the recurrent neural network architecture called the *SERNN* and its fast, accurate and robust training method called the MRLS algorithm. This chapter investigates how these methods perform for the identification of a dynamic nonlinear system. One way to evaluate this performance is known as *simulation*. Simulation is a very useful tool to investigating new on-line identification algorithms especially for nonlinear and time-varying systems. Many analyses on the SRLS clones have been carried out based on assumptions about input vector, linear time-invariant system, and zero-mean residual error. However, the numerical problem, for example, *burst phenomenon* on the SRLS algorithm, is still observed frequently for a variety of nonlinear systems' identification. In this chapter, the performance of the proposed on-line system identification method developed in this thesis will be shown by comparing it to conventional methods. This performance improvement can be described in terms of the tracking speed, the accuracy of the identification error and the robustness with respect to a variety of situations between the two approaches. The two methods are listed as follows:

1. Conventional Methods:
    - Neural Architecture: Two-layer RNN
    - Training Algorithm: SRLS Algorithm
    - Features: application of the SRLS algorithm to hidden-layer training based on L_2-norm optimization technique (novel); exponential-weight constant forgetting factor (EW-CFF); linearization by the inverse of activation function; periodic updating of covariance update law to keep *alertness*.
2. Proposed Methods (developed in this thesis):
    - Neural Architecture: Two-layer SERNN (**novel**)
    - Training Algorithm: MRLS Algorithm (**novel**)
    - Features: variable bias by error recurrent (**novel**); supervision recurrent (**novel**); application of MRLS algorithm to hidden-layer training based on L_2-norm optimization technique (**novel**); measurement based exponential-weight variable forgetting factor (**novel**); linearization by the inverse of the activation function; periodic updating of covariance update law with EW-VFF to keep *alertness* all the time.

These algorithms were applied to four nonlinear SISO plants and one nonlinear MIMO system under four different initial weight values for diverse simulation scenarios. The plant dynamics and simulation results are shown in the following sections where details are given through performance tables and graphs.

6.2 Test On A Nonlinear SISO Plant 1

6.2.1 Dynamics Of Plant 1

Input of Plant1:

if ($k \leqq 100$)

$u(k) = 0.5 \cdot \cos(2\pi \cdot f_1 \cdot kT_s/5.0) - 0.8 \cdot \sin(2\pi \cdot f_1 \cdot kT_s/30.0) + 0.05$

elseif ($100 < k \leqq 250$)

$u(k) = 0.2 \cdot kT_s - 1.2$

elseif ($250 < k \leqq 460$)

$u(k) = -0.05 \cdot kT_s + 0.4$

elseif ($460 < k$)

$u(k) = 0.6 \cdot \sin(2\pi \cdot kT_s/120.0) + 0.3 \cdot \cos(2\pi \cdot f_2 \cdot kT_s/45.0)$

end

where $f_1 = 90.0[\text{Hz}], f_2 = 100.0[\text{Hz}]$ and $T_s = 0.01[\text{sec}]$

Output of Plant1:

$$\begin{aligned} y^d(k) = {} & 0.25 \cdot y^d(k-1)^2 + 0.2 \cdot y^d(k-1) + 0.2 \cdot y^d(k-2) + \\ & 0.45 \cdot \sin\Big(0.5 \cdot (y^d(k-1) + y^d(k-2)\Big) \times \cos\Big(0.4 \cdot (y^d(k-1) + \\ & y^d(k-2)\Big) + 0.3 \cdot u(k-1) - 0.01 \end{aligned}$$

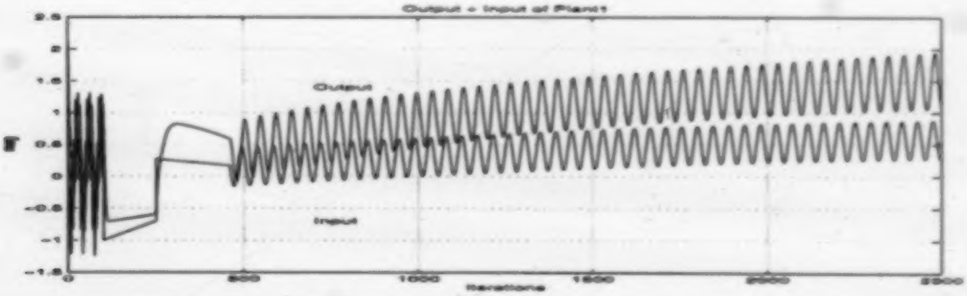


Figure 60: Input and Output of Nonlinear Plant1

6.2.2 Performance Table Of Identification On Plant 1

Application to Nonlinear **Plant1**				
Using **Conventional** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	Wind-up	Wind-up	Wind-up (Frequently)
Initial Tracking	Good	Not Bad	Not Bad	Good
Error Margin of Steady State (< 0.1)	Good	**Divergent**	**Divergent**	**Divergent**
Trace of P with Periodic Resetting	Alert	Alert	Alert	Poorly Alert
Sum of Q Elements	None Exists	None Exists	None Exists	None Exists

Table 6.1: Identification Performance of **Conventional** Methods on **Plant 1**

Application to Nonlinear **Plant1**				
Using **Proposed** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	O.K.	O.K. (Always)
Initial Tracking	Good	Good	Good	Good
Error Margin of Steady State (< 0.1)	Excellent	Excellent	Excellent	Excellent
Trace of P with Periodic Resetting	Alert	Alert	Alert	Alert
Sum of Q Elements	Alert	Alert	Alert	Alert

Table 6.2: Identification Performance of **Proposed** Methods on **Plant 1**

6.3 Test On A Nonlinear SISO Plant 2

6.3.1 Dynamics Of Plant 2

Input of Plant2:

if ($k \leqq 250$)

$$u(k) = 0.5 \cdot \cos(2\pi \cdot f_1 \cdot kT_s/50.0) - 0.8 \cdot \sin(2\pi \cdot f_1 \cdot kT_s/30.0) + 0.075$$

elseif ($250 < k \leqq 380$)

$$u(k) = 0.2 \cdot kT_s - 1.2$$

elseif ($380 < k \leqq 560$)

$$u(k) = -0.05 \cdot kT_s + 0.4$$

elseif ($560 < k$)

$$u(k) = 0.3 \cdot \sin(2\pi \cdot f_1 \cdot kT_s/120.0) + 0.3 \cdot \cos(2\pi \cdot f_2 \cdot kT_s/45.0)$$

end

where $f_1 = 70.0$[Hz],$f_2 = 10.0$[Hz] and $T_s = 0.01$[sec]

Output of Plant2:

$$\begin{aligned} y^d(k) = {} & 0.25 \cdot y^d(k-1)^2 - 0.2 \cdot y^d(k-1) + 0.45 \cdot \sin\Big(0.5 \cdot (y^d(k-1)\Big) \\ & \times \cos\Big(0.4 \cdot (y^d(k-1)\Big) + 0.3 \cdot u(k-1) - 0.02 \end{aligned}$$

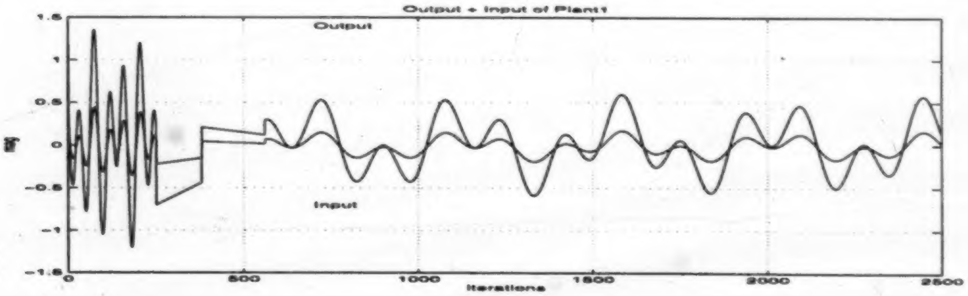


Figure 61: Input and Output of Nonlinear Plant2

6.3.2 Performance Table Of Identification On Plant 2

Application to Nonlinear **Plant2**				
Using **Conventional** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	Wind-up	Wind-up (Often)
Initial Tracking	Not Bad	Good	Not Bad	Not Good
Error Margin of Steady State (< 0.1)	Excellent	Excellent	**Divergent**	**Divergent**
Trace of P with Periodic Resetting	Alert	Poorly Alert	Not Alert	Poorly Alert
Sum of Q Elements	None Exists	None Exists	None Exists	None Exists

Table 6.3: Identification Performance of **Conventional** Methods on **Plant 2**

Application to Nonlinear **Plant2**				
Using **Proposed** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	O.K.	O.K. (Always)
Initial Tracking	Good	Good	Good	Good
Error Margin (< 0.1)	Excellent	Excellent	Excellent	Excellent
Trace of P with Periodic Resetting	Alert	Alert	Alert	Alert
Sum of Q Elements	Alert	Alert	Alert	Alert

Table 6.4: Identification Performance of **Proposed** Methods on **Plant 2**

6.4 Test On A Nonlinear SISO Plant 3

6.4.1 Dynamics Of Plant 3

Input of Plant3:

if ($k \leqq 100$)

$u(k) = 0.3 \cdot \cos(2\pi \cdot f_1 \cdot kT_s/25.0) - 0.5 \cdot \sin(2\pi \cdot f_1 \cdot kT_s/70.0)$

elseif ($100 < k \leqq 200$)

$u(k) = 0.005 \cdot kT_s - 0.3$

elseif ($200 < k \leqq 300$)

$u(k) = -0.005 \cdot kT_s + 1.72$

elseif ($300 < k$)

$u(k) = 0.4 \cdot \sin(2\pi \cdot f_1 \cdot kT_s/120.0) + 0.3 \cdot \cos(2\pi \cdot f_2 \cdot kT_s/45.0)$

end

where $f_1 = 1.0[\mathrm{Hz}], f_2 = 1.0[\mathrm{Hz}]$ and $T_s = 1.0[\mathrm{sec}]$

Output of Plant3:

$$
\begin{aligned}
y^d(k) = 0.2 \cdot y^d(k-1)^2 - 0.2 \cdot y^d(k-1) + 0.25 \cdot y^d(k-2) + \\
0.45 \cdot \sin\Big(0.5 \cdot (y^d(k-1) + y^d(k-2))\Big) \times \cos\Big(0.5 \cdot (y^d(k-1) + \\
y^d(k-2))\Big) + 0.25 \cdot u(k-1)
\end{aligned}
$$

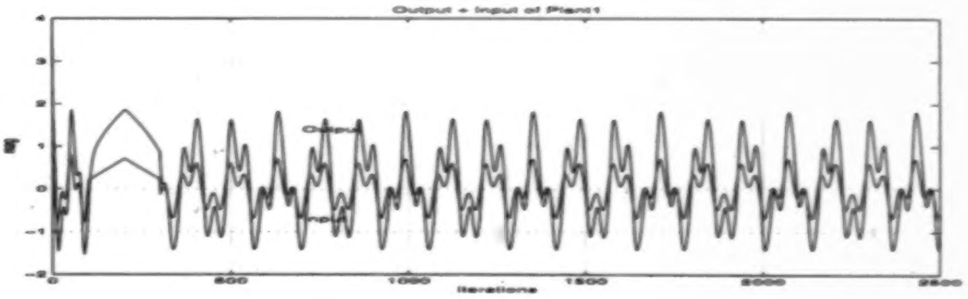


Figure 62: Input and Output of Nonlinear Plant3

6.4.2 Performance Table Of Identification On Plant 3

Application to Nonlinear **Plant3**				
Using **Conventional** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	Wind-up	Wind-up (Frequently)
Initial Tracking	Bad	Bad	Not Bad	Bad
Error Margin of Steady State (< 0.1)	Bad	Bad	**Divergent**	**Divergent**
Trace of P with Periodic Resetting	Alert	Poorly Alert	Poorly Alert	Poorly Alert
Sum of Q Elements	None Exists	None Exists	None Exists	None Exists

Table 6.5: Identification Performance of **Conventional** Methods on **Plant 3**

Application to Nonlinear Plant3				
Using **Proposed** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	O.K.	O.K. (Always)
Initial Tracking	Good	Good	Good	Good
Error Margin (< 0.1)	Excellent	Excellent	Excellent	Good
Trace of P with Periodic Resetting	Alert	Alert	Alert	Alert
Sum of Q Elements	Alert	Alert	Alert	Alert

Table 6.6: Identification Performance of **Proposed** Methods on Plant 3

6.5 Test On A Nonlinear SISO Plant 4

6.5.1 Dynamics Of Plant 4

Input of Plant4:

$$u(k) = \Big(0.3 \cdot \cos(2\pi k/300.0) - 0.05 \cdot \sin(2\pi k/120.0)\Big) \times 1.5$$

Output of Plant4:

$$y^d(k) = \Big(0.2 \cdot \exp\big(u(k) + \sin(10.0 \cdot u(k))\big) + 0.9 \cdot \sin\big(u(k) + 20.0 \cdot \exp(0.5 \cdot u(k))\big)\Big) \times 1.5$$

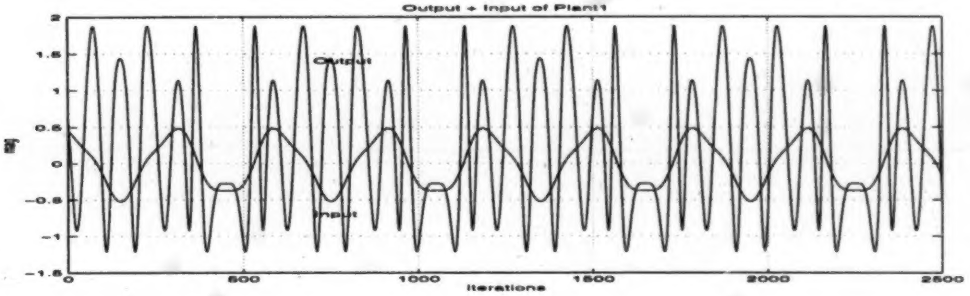


Figure 63: Input and Output of Nonlinear Plant4

6.5.2 Performance Table Of Identification On Plant 4

Application to Nonlinear **Plant4**				
Using **Conventional** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	Wind-up	Wind-up	Wind-up(Frequently)
Initial Tracking	Very Bad	Good	Not Bad	Very Bad
Error Margin of Steady State (< 0.1)	Very Bad	**Divergent**	**Divergent**	**Divergent**
Trace of P with Periodic Resetting	Poorly Alert	Poorly Alert	Poorly Alert	Poorly Alert
Sum of Q Elements	None Exists	None Exists	None Exists	None Exists

Table 6.7: Identification Performance of **Conventional** Methods on **Plant 4**

Application to Nonlinear **Plant4**				
Using **Proposed** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	O.K.	O.K. (Always)
Initial Tracking	Good	Better	Good	Good
Error Margin (< 0.1)	Good	Good	Good	Good
Trace of P with Periodic Resetting	Alert	Alert	Alert	Alert
Sum of Q Elements	Alert	Alert	Alert	Alert

Table 6.8: Identification Performance of **Proposed** Methods on **Plant 4**

6.6 Test On A Nonlinear MIMO Plant 5 Of Swing Leg

6.6.1 Dynamics Of Plant 5

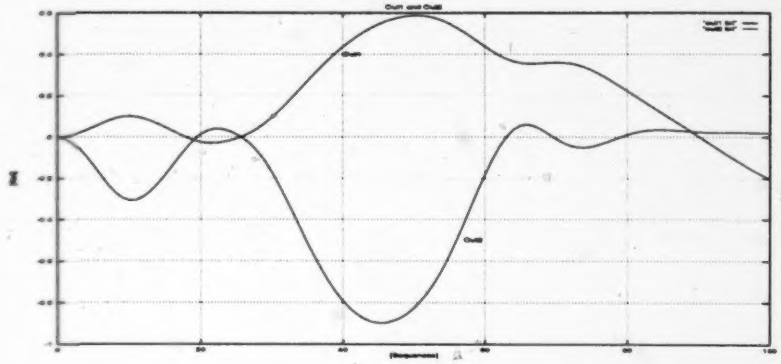


Figure 64: Two Outputs of Nonlinear MIMO Plant 5 of Swing Leg

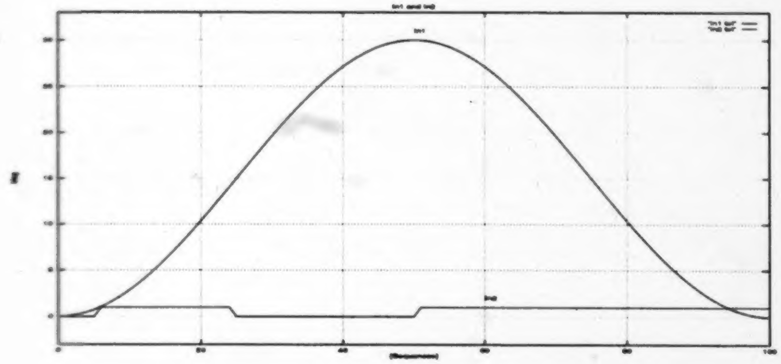


Figure 65: Two Inputs of Nonlinear MIMO Plant 5 of Swing Leg

The dynamics profiles of two-input and two-output are shown in Figures 64 and 65 for the swinging leg [2] being identified. This is a two dimensional, two-segment compound pendulum with the *hip joint* fixed in an inertial frame of reference. The *ankle joint* is assumed to be fused at a right angle. Nonlinear elastic reactions were applied to the joints to simulate tendon activity at extreme angles and linear damping was applied at both joints. Body parameters for an average person with 65 Kg weight and 170 cm height were used [3]. The swing leg was actuated by two means:

- The first was an external motor torque applied at the hip joint (input1).
- The second was artificial stimulation of the quadriceps muscles creating an internal torque at the knee joint (input2). The muscle stimulation is represented by a normalized value between 0 (no stimulation) and 1 (full stimulation).

Given these two input signals, the swing leg model returns two output signals: the hip (out1) and knee (out2) angles. The fast, accurate and robust identification techniques for above neuromuscular system can play a crucial role to develop a powered hybrid FES (functional electrical stimulation) orthosis for paraplegics[1].

---

[1] By W.-K. Son, S. Madan, A. Thrasher and B.J. Andrews
Fast Neural System Identification: Application to Rehabilitation Engineering published in 1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, vol.2, pp.539–542, August 20–22, Victoria, B.C., Canada,1997.

6.6.2 Performance Table Of Identification On Plant 5

Application to Nonlinear **Plant5**				
Using **Conventional** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	Wind-up	Wind-up	Wind-up(Frequently)
Initial Tracking	Excellent	Excellent	Excellent	Excellent
Error Margin of Steady State (< 0.1)	Bad	**Divergent**	**Divergent**	**Divergent**
Trace of P with Periodic Resetting	Poorly Alert	Poorly Alert	Poorly Alert	Alert
Sum of Q Elements	None Exists	None Exists	None Exists	None Exists

Table 6.9: Identification Performance of **Conventional** Methods on **Plant 5**

Application to Nonlinear **Plant 5**				
Using **Proposed** Architecture & Training				
Initial Weight Sets	I	II	III	IV(Random No.)
Identifiability	O.K.	O.K.	O.K.	O.K. (Always)
Initial Tracking	Excellent	Excellent	Excellent	Excellent
Error Margin of Steady State (< 0.1)	Excellent	Good	Excellent	Excellent
Trace of P with Periodic Resetting	Alert	Alert	Alert	Alert
Sum of Q Elements	Alert	Alert	Alert	Alert

Table 6.10: Identification Performance of **Proposed** Methods on **Plant 5**

6.7 Conclusions and Simulation Results

The computer simulation results of the on-line system identification have been presented to show the outstanding performance of the proposed methodologies compared to the existing methods when used for the same purpose. All performance comparisons have been summarized in the shown tables. In this research, the application of the RLS algorithm to training of *multi-layer* neural networks has been successfully tried for the first time, to the author's best knowledge. In order for the conventional RLS algorithm to be used for the multi-layer neural nets training, the following problems should have been solved for its direct application.

- The RLS can be applied only to the *linear parametric* model. But ANNs are not.
- The RLS requires a *clear supervisory signal* to train the desired layer. But the hidden layer does not have an explicit teaching signal. The RLS cannot be used for hidden layer training. That is why the BP algorithm uses the error backpropagation principle for the hidden layer training even though there exist lots of drawbacks. The combination of the Kalman filter and the BP algorithm are sometimes used for the output and hidden layer training respectively to circumvent the *absence of teaching signal* in the hidden layer. However, this combinational method still has the drawbacks of the BP algorithm.
- The RLS has the **crucial heel-of-Achilles of numerical burst phenomenon**. This makes the use of conventional RLS for neural training difficult. The numerically related problems in the RLS are listed below:
  - The RLS is also very sensitive to the *initial estimates*. For example, when the *random numbers* are used for initialization, it frequently diverges during the identification.
  - The RLS is not robust with respect to the *periodical resetting of covariance matrix* to keep the estimation gain *alert* under poor *persistent excitation*

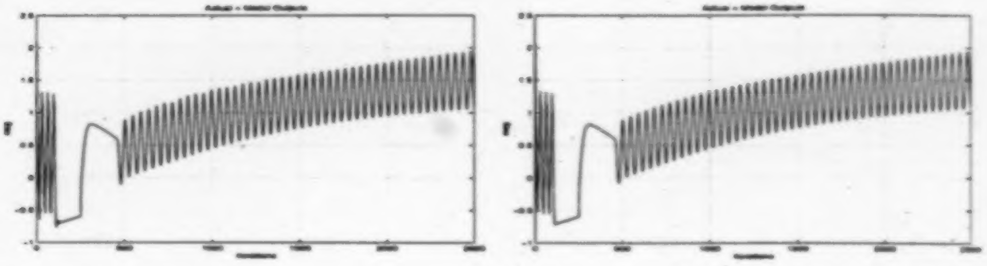
conditions. It may lack the capability of recovering the lost information by the resetting method, and thereby it diverges.

- As a fast weights training algorithm, the RLS performance can easily deteriorate during *initial tracking*, highly dependent on initial weights and the system being considered. Thus the RLS is not robust.

Therefore, **the direct application of the standard RLS algorithm to neural networks is not adequate**. All aforementioned problems have been successfully solved and/or improved by the proposed SERNN neural architecture and MRLS training algorithm which incorporate several novel features. The performance tables shown in the previous section and the detailed simulation graphs in the following section **clearly demonstrate the superior performance of developed algorithms for the purpose of on-line system identification**. The SERNN architecture and MRLS algorithm show fast, accurate and robust performance for on-line neural system identification. The *speed* is shown by the prompt transient-behavior at the beginning of training and for sudden changes in the input. The *accuracy* represents a small identification error in the steady state, and the *robustness* by the sure-identifiability under diverse situations without changing any of the adjustable free-parameters in the proposed algorithms. Although these results may not be used to generalize its superior performance for other systems not shown here, the proposed methods clearly show the solution/improvement over the existing methods of the RLS clone in conjunction with neural nets training. It is also clear that the developed algorithms would at least promise *better-than or equal* performance for other unknown nonlinear systems. The detailed simulation results are presented in the next subsections.

The proposed training algorithm and neural architecture have lots of potential for the excellent design of the adaptive control by their powerful capability of on-line system identification. For the design purpose here, the controller should not be an *ad-hoc* device for the specific plant. The ad-hoc controller becomes useless when different plants are considered or the dynamic uncertainty is serious for a certain environment. As a matter of fact, the design of an ideal *universal* nonlinear controller should not use any pre-knowledge about the plant except for few input/output measurements. The robust system identification methodology developed in this thesis may suggest intuitions and possible solutions for this universal control concept. These techniques use only I/O measurements and do not adjust the free-parameters for a specific system under consideration. The performance of the proposed system identification algorithms can be said to satisfy these requirements to a large extent. The next chapter presents the design of an optimal (self-tuning) adaptive control which is heavily dependent on the performance of the on-line neural system identification based on the *certainty-equivalence* principle.

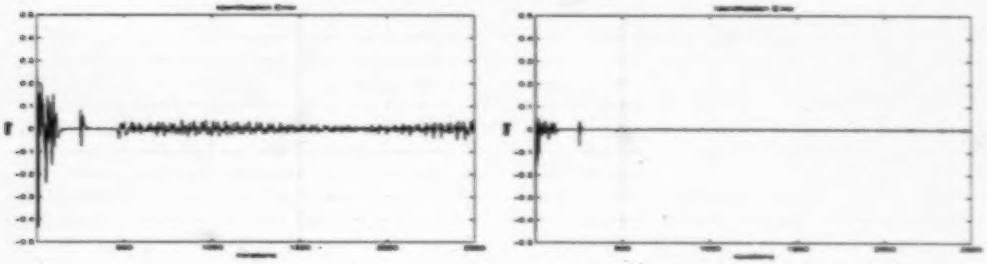
6.7.1 With Initial Weight Set 1 On Plant 1



(a) By conventional method

(b) By proposed method

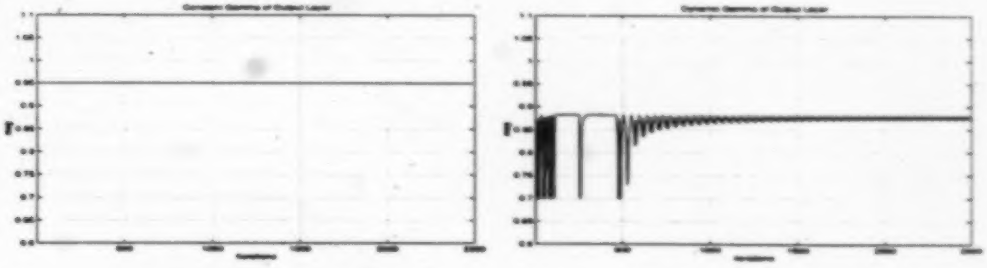
Figure 66: Actual output **identification** by model output



(a) By conventional method

(b) By proposed method

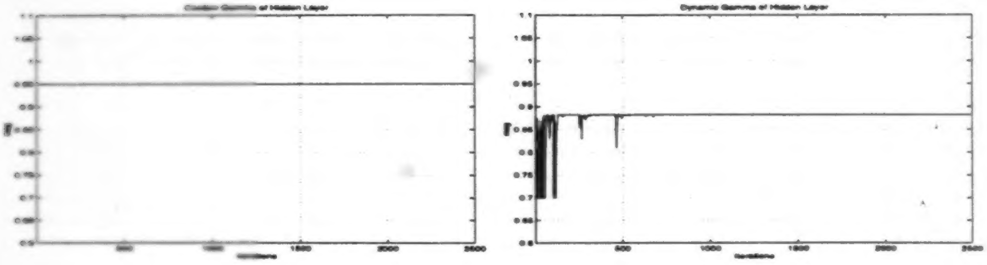
Figure 67: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.95

(b) Proposed VFF

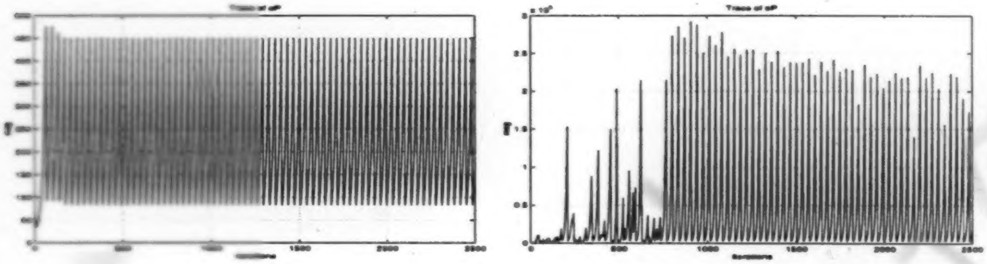
Figure 68: Constant and Variable **Forgetting Factors** in **output layer**



(a) Conventional CFF= 0.95

(b) Proposed VFF

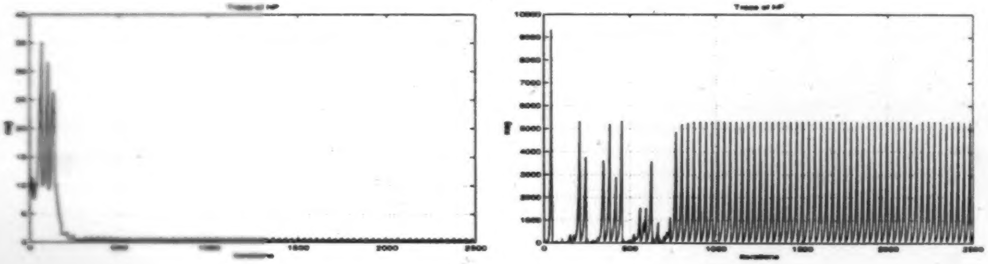
Figure 69: Constant and Variable **Forgetting Factors** in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

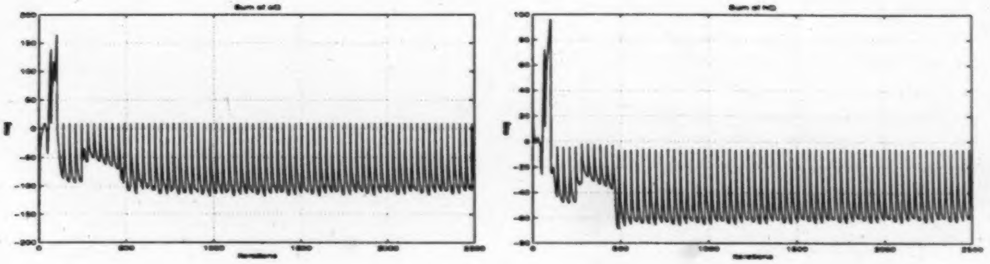
Figure 70: **Trace of covariance matrices** in **output layer**



(a) conventional method

(b) proposed method

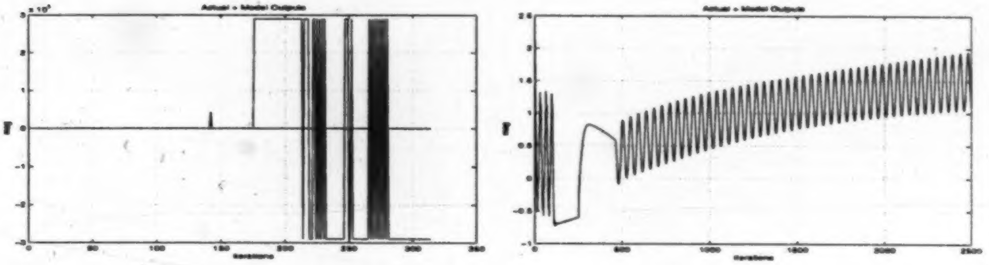
Figure 71: **Trace of covariance matrices** in **hidden layer**



(a) For proposed method in output layer (b) For proposed method in hidden layer

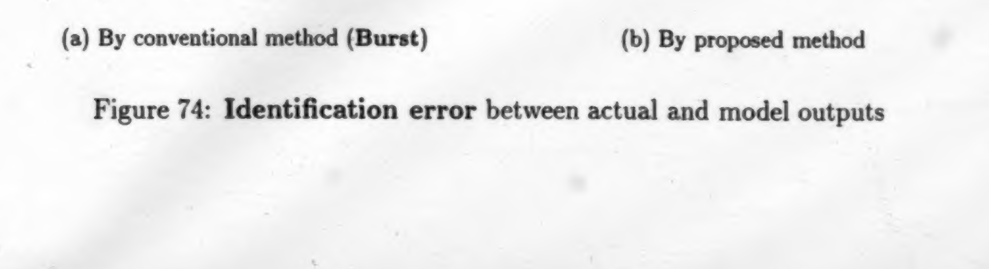
Figure 72: Sum of Q Elements

6.7.2 With Initial Weight Set 2 On Plant 1



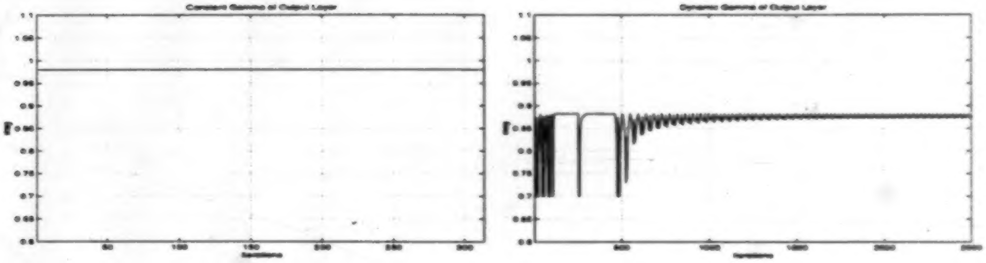
(a) By conventional method (**Wind-up**) (b) By proposed method

Figure 73: Actual output **identification** by model output



(a) By conventional method (**Burst**) (b) By proposed method

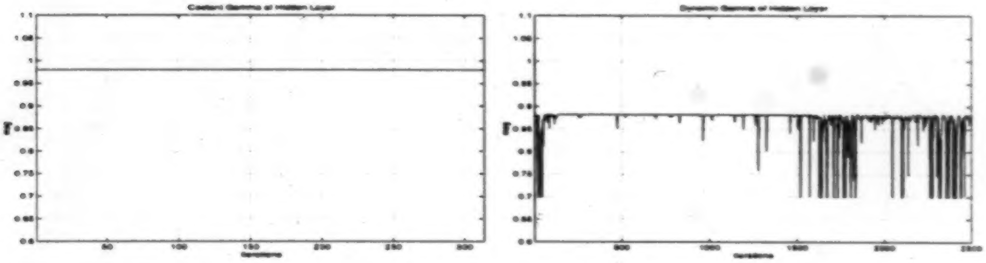
Figure 74: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

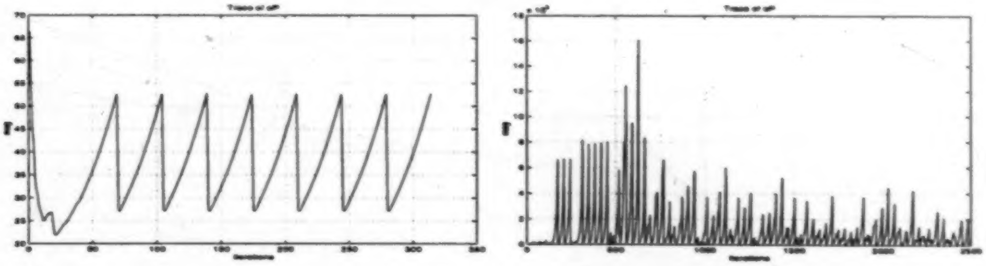
Figure 75: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

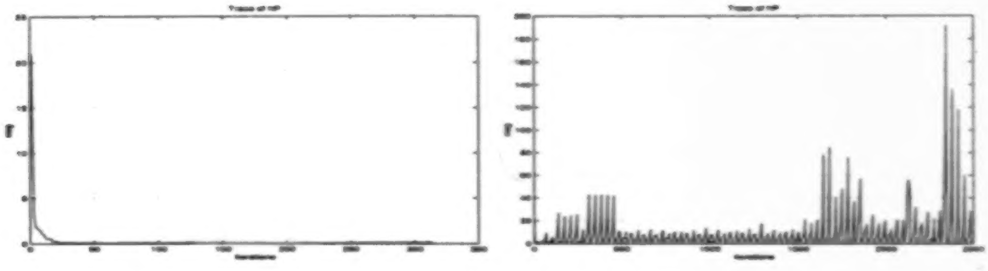
Figure 76: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

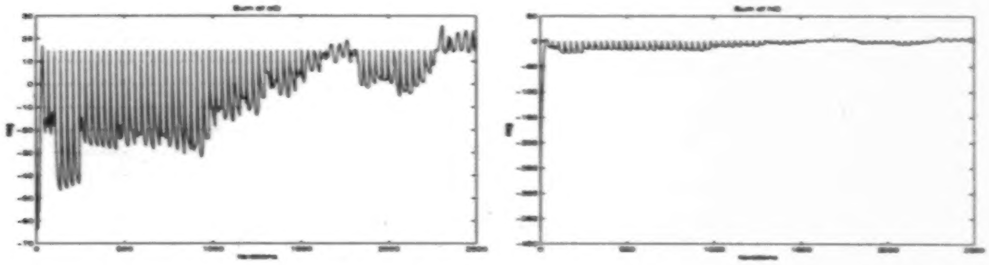
(b) Periodic resetting with VFF

Figure 77: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Not alert**)   (b) Periodic resetting with VFF

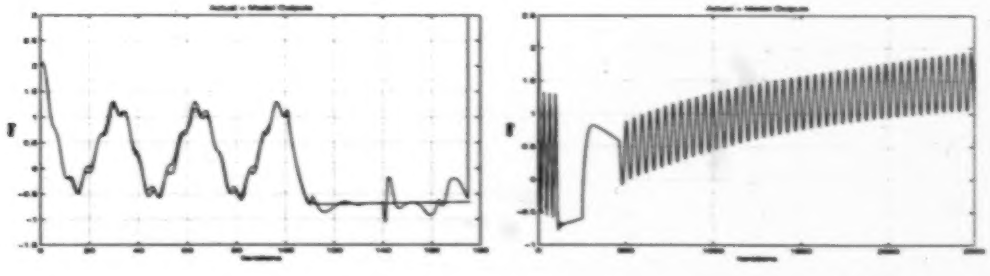
Figure 78: Trace of covariance matrices in hidden layer



(a) For proposed method in output layer   (b) For proposed method in hidden layer

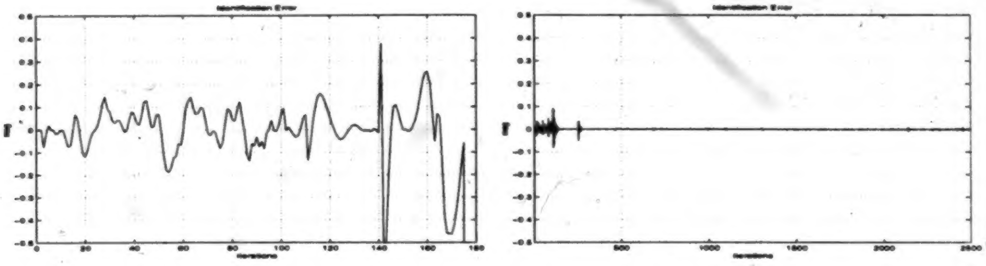
Figure 79: Sum of Q Elements

6.7.3 With Initial Weight Set 3 On Plant 1



(a) By conventional method (**Wind-up**)   (b) By proposed method

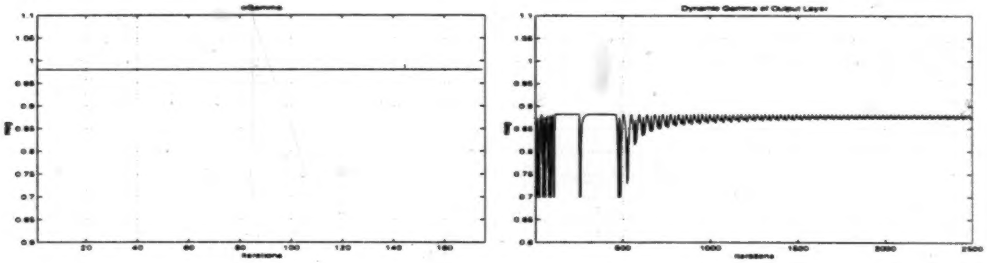
Figure 80: Actual output **identification** by model output



(a) By conventional method (**Burst**)

(b) By proposed method

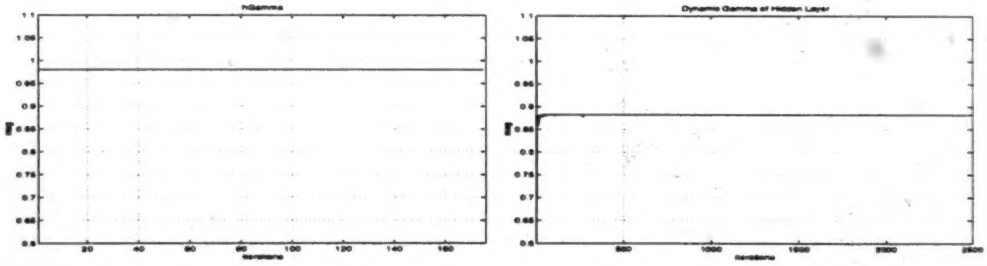
Figure 81: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

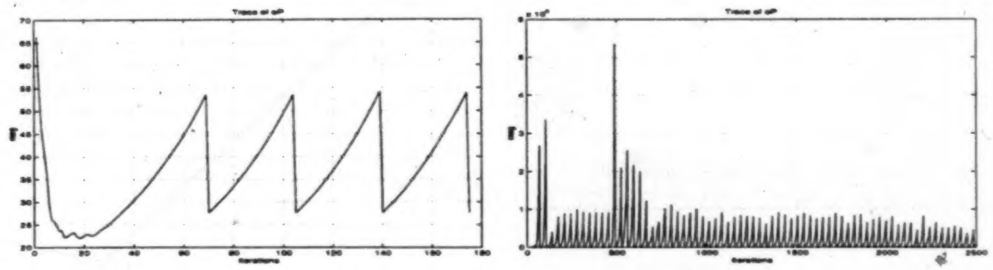
Figure 82: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

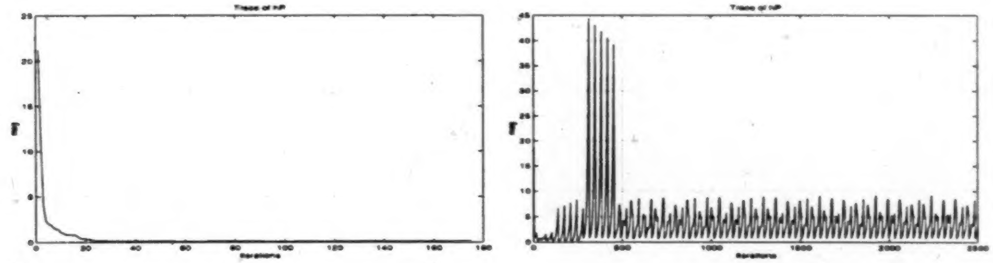
Figure 83: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

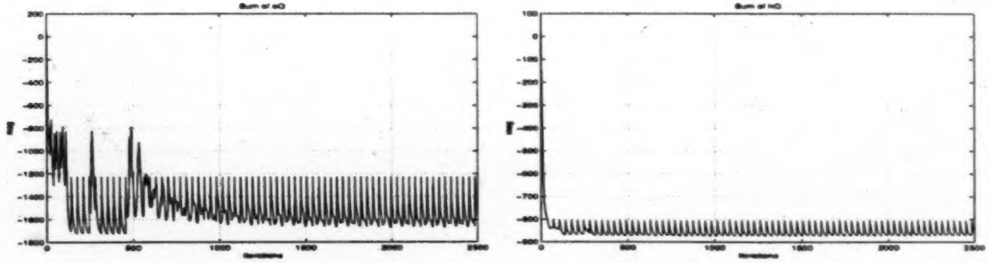
Figure 84: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

Figure 85: Trace of covariance matrices in hidden layer

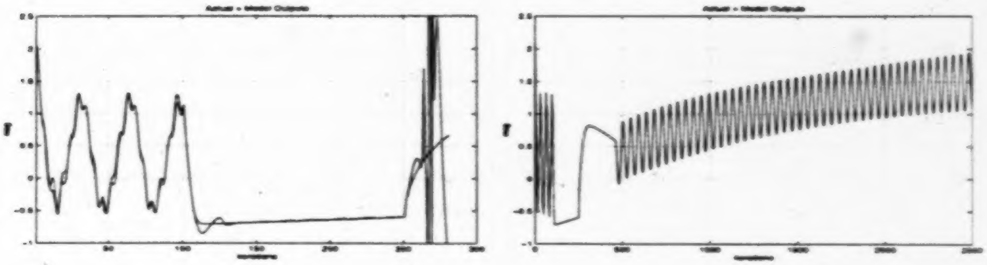


(a) For proposed method in output layer

(b) For proposed method in hidden layer

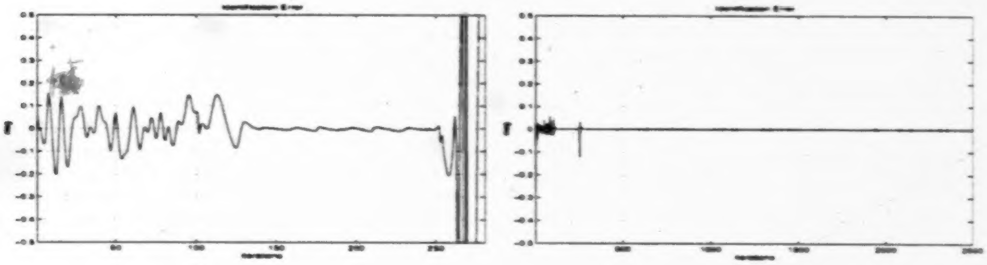
Figure 86: Sum of Q Elements

6.7.4 With Initial Weight Set 4 (Random Number) On Plant 1



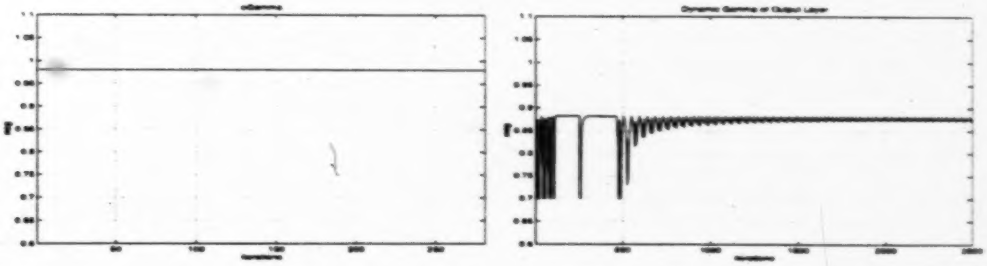
(a) By conventional method (**Wind-up**) (b) By proposed method

Figure 87: Actual output **identification** by model output



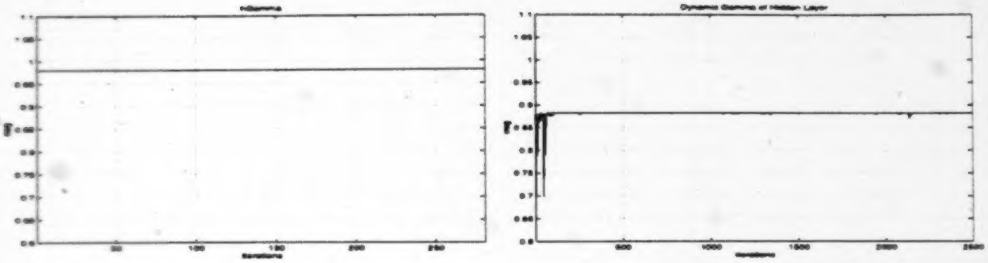
(a) By conventional method (**Burst**) (b) By proposed method

Figure 88: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98 (b) Proposed VFF

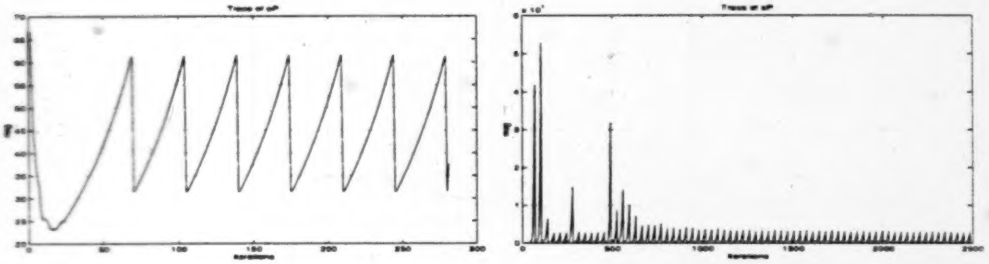
Figure 89: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

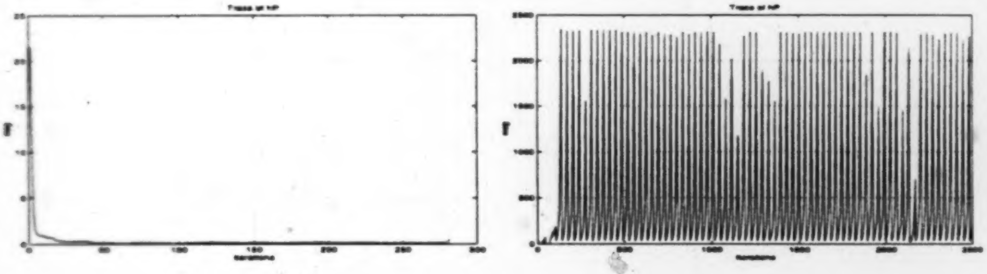
Figure 90: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

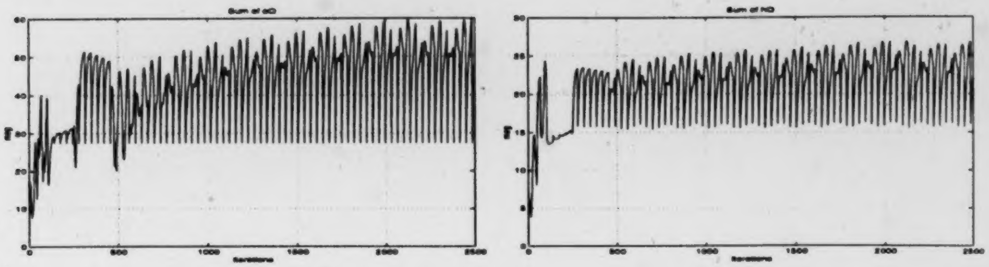
Figure 91: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

Figure 92: Trace of covariance matrices in hidden layer

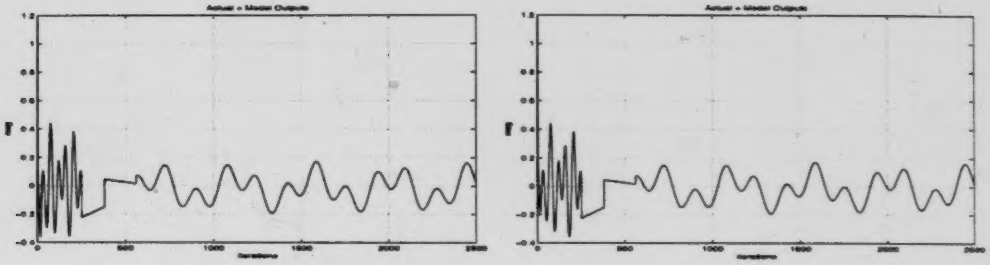


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 93: Sum of Q Elements

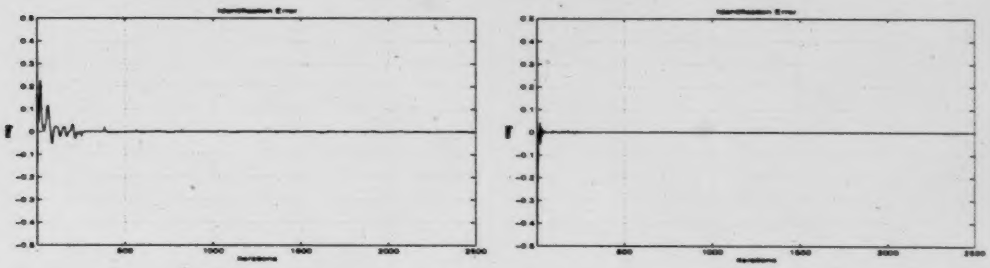
6.7.5 With Initial Weight Set 1 On Plant 2



(a) By conventional method

(b) By proposed method

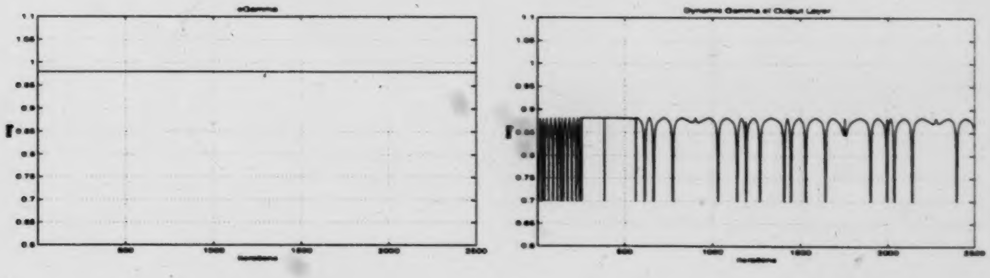
Figure 94: Actual output **identification** by model output



(a) By conventional method

(b) By proposed method

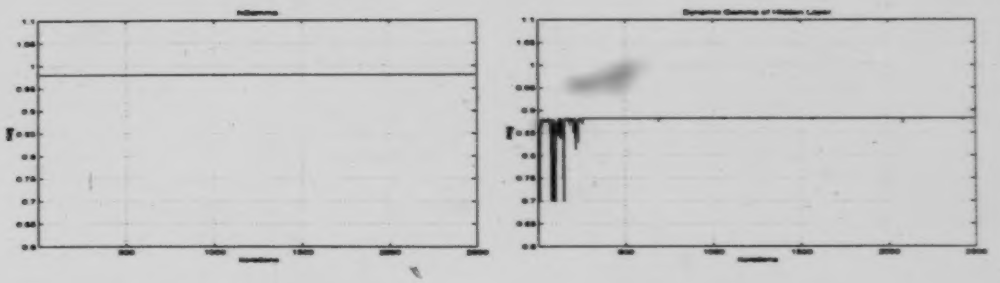
Figure 95: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

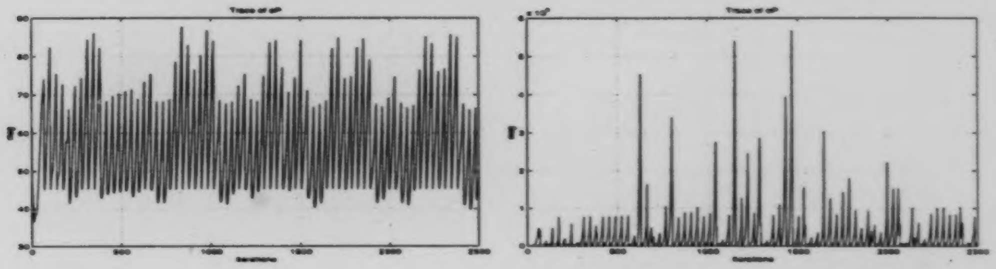
Figure 96: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

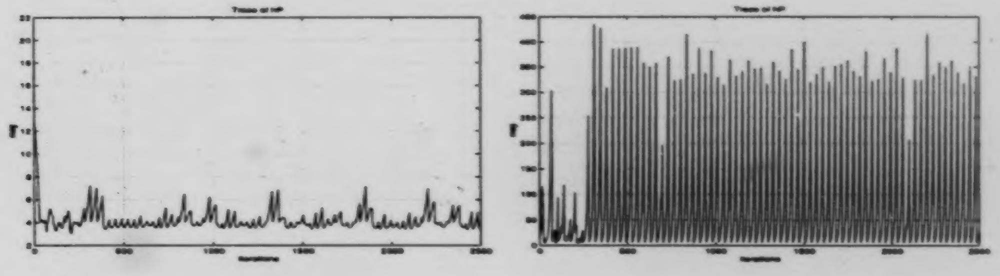
Figure 97: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

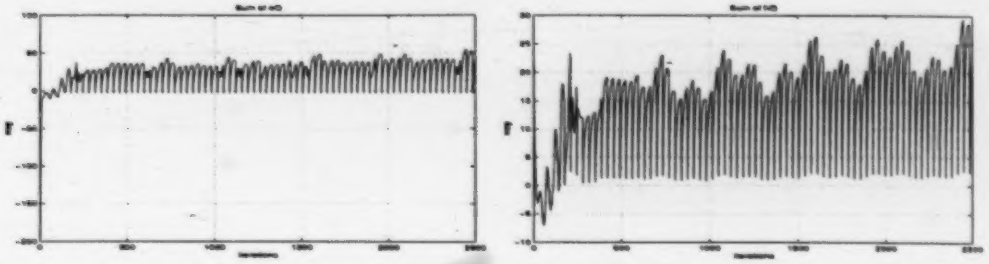
Figure 98: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

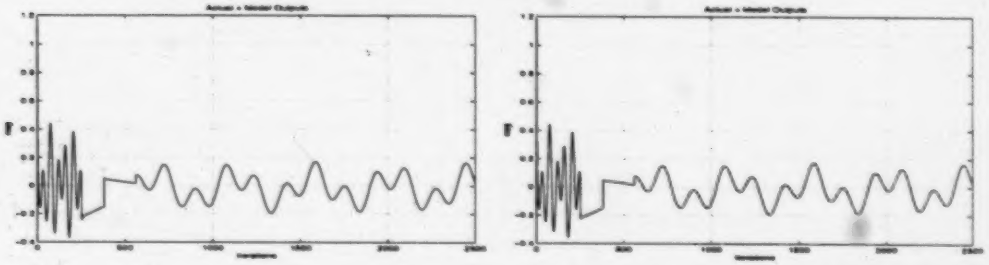
Figure 99: Trace of covariance matrices in hidden layer



(a) For proposed method in output layer (b) For proposed method in hidden layer

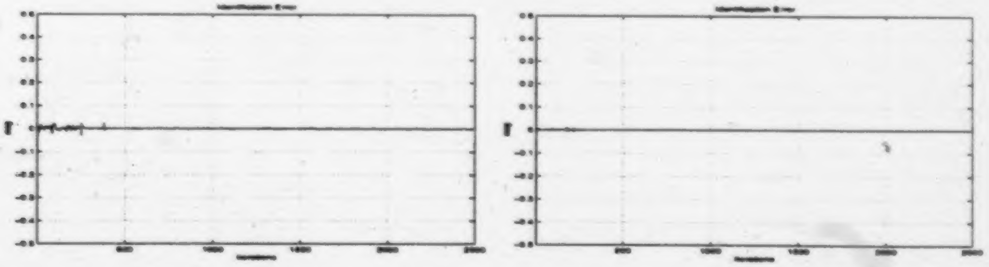
Figure 100: Sum of Q Elements

6.7.6 With Initial Weight Set 2 On Plant 2



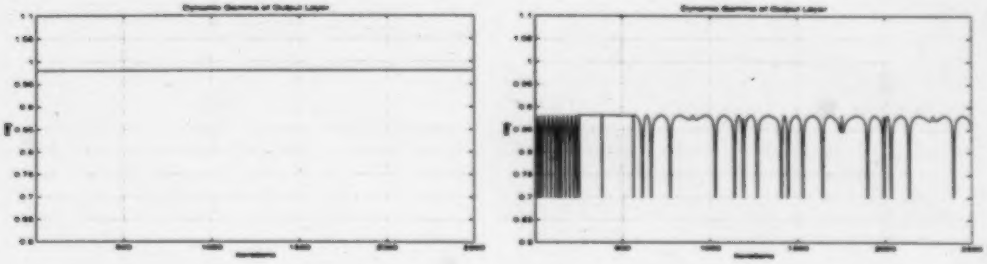
(a) By conventional method (b) By proposed method

Figure 101: Actual output **identification** by model output



(a) By conventional method (b) By proposed method

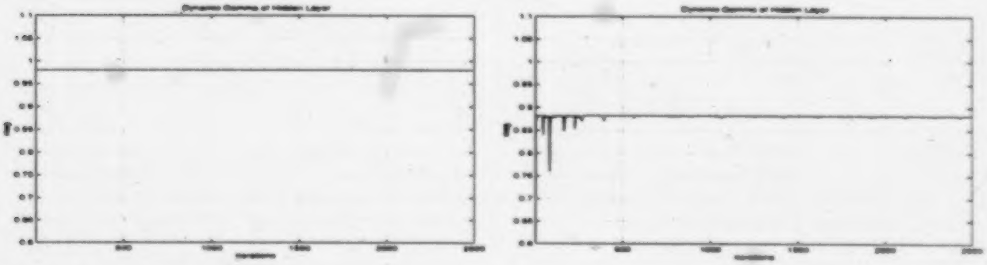
Figure 102: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

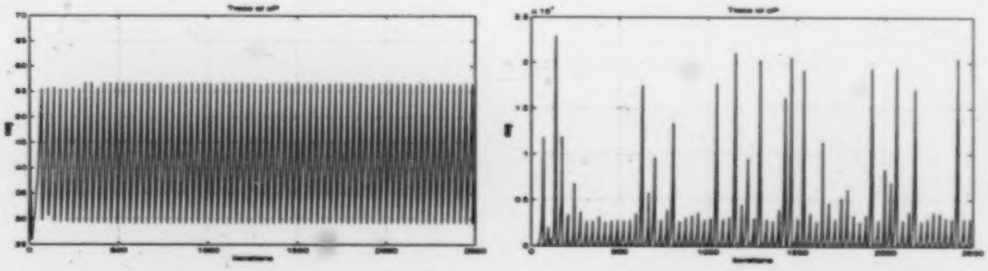
Figure 103: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

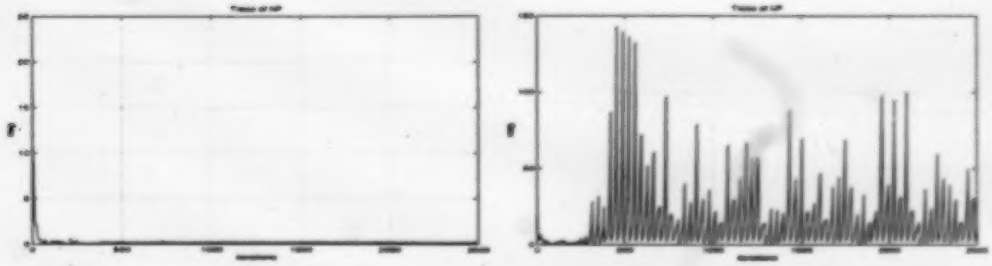
Figure 104: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

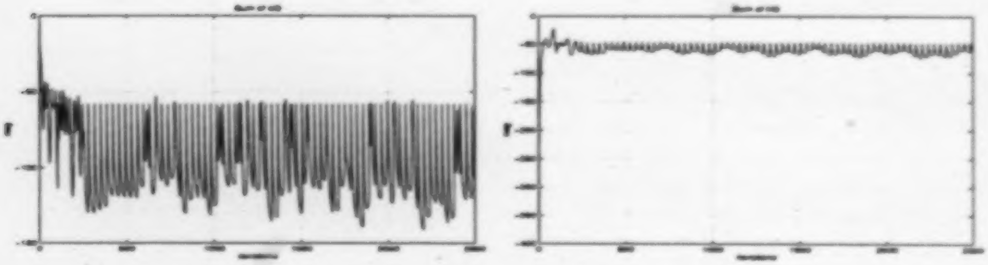
Figure 105: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Poorly alert**)

(b) Periodic resetting with VFF

Figure 106: Trace of covariance matrices in hidden layer

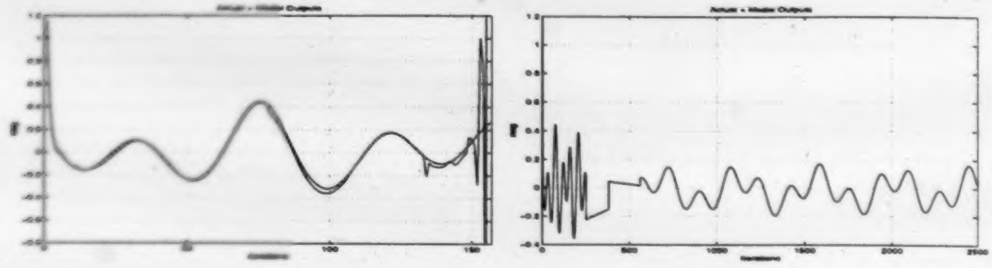


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 107: Sum of Q Elements

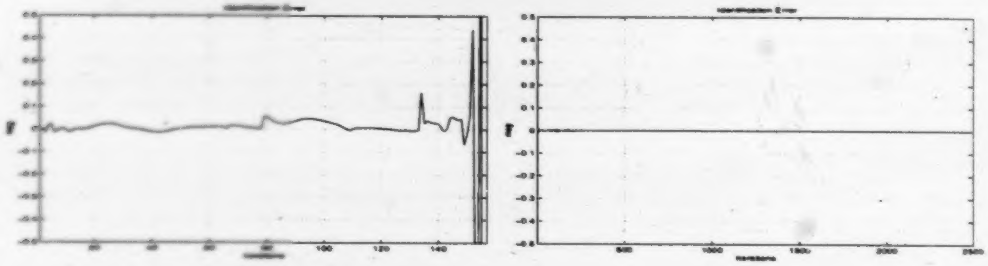
6.7.7 With Initial Weight Set 3 On Plant 2



(a) By conventional method (**Wind-up**)

(b) By proposed method

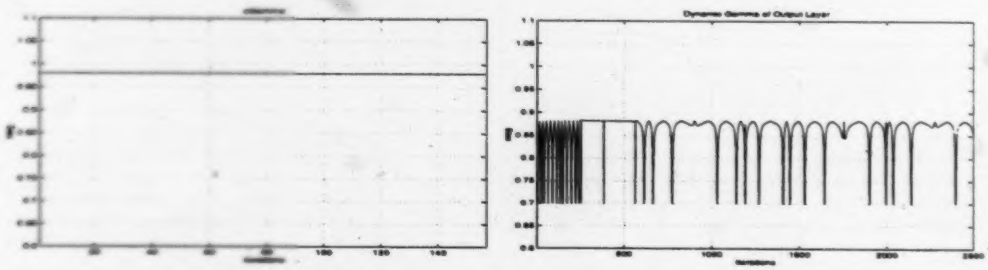
Figure 108: Actual output **identification** by model output



(a) By conventional method (**Burst**)

(b) By proposed method

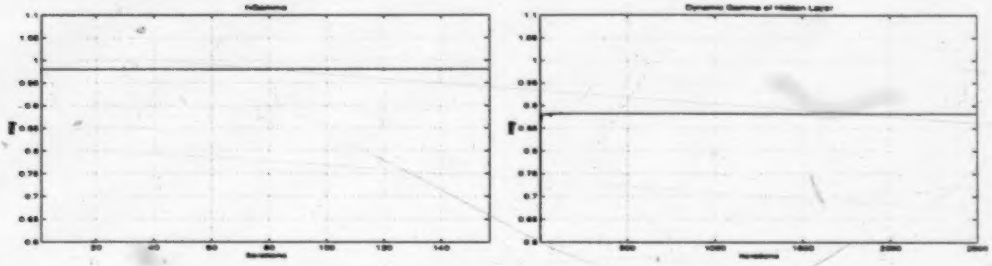
Figure 109: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

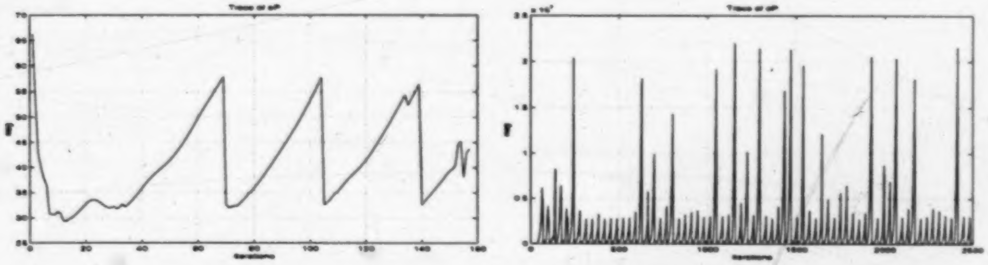
Figure 110: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

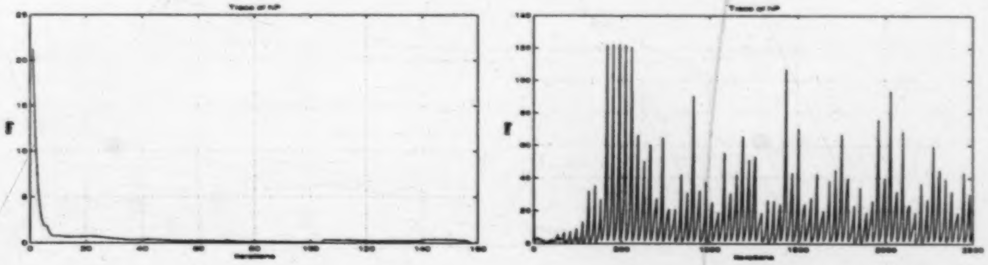
Figure 111: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

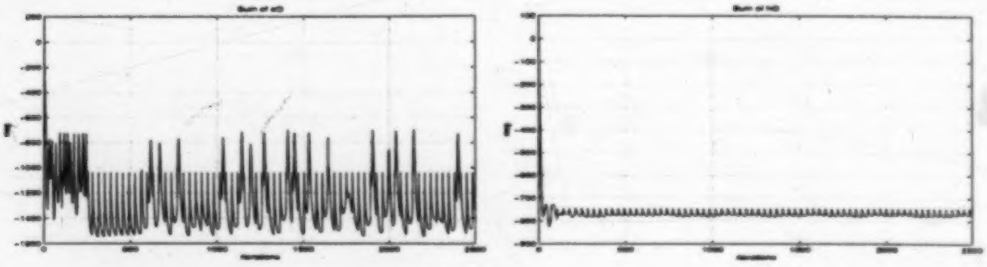
Figure 112: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (Poorly alert)

(b) Periodic resetting with VFF

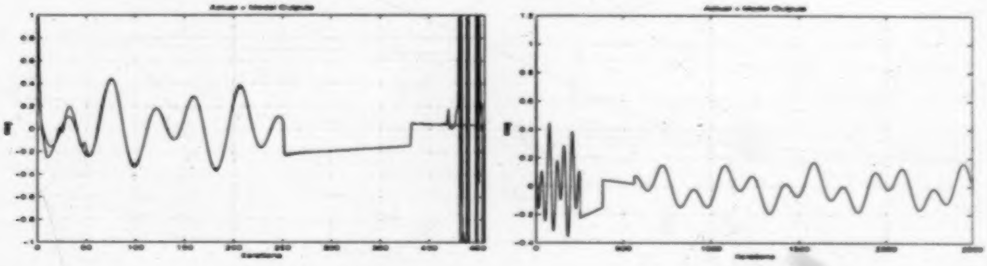
Figure 113: Trace of covariance matrices in hidden layer



(a) For proposed method in output layer (b) For proposed method in hidden layer

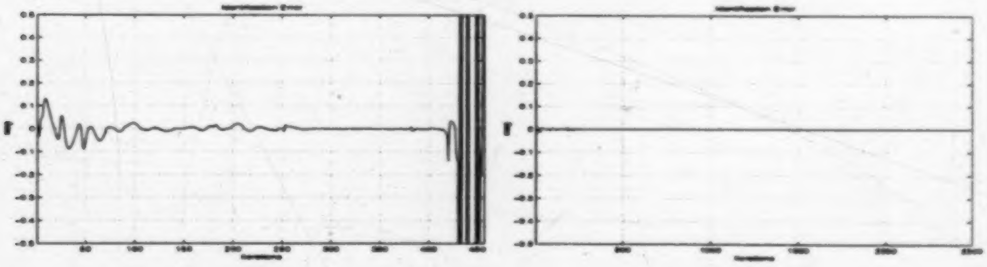
Figure 114: Sum of Q Elements

6.7.8 With Initial Weight Set 4 On Plant 2



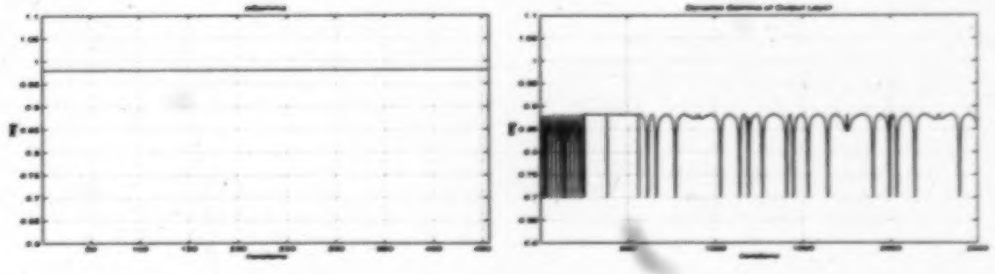
(a) By conventional method (**Wind-up**) (b) By proposed method

Figure 115: Actual output **identification** by model output



(a) By conventional method (**Burst**) (b) By proposed method

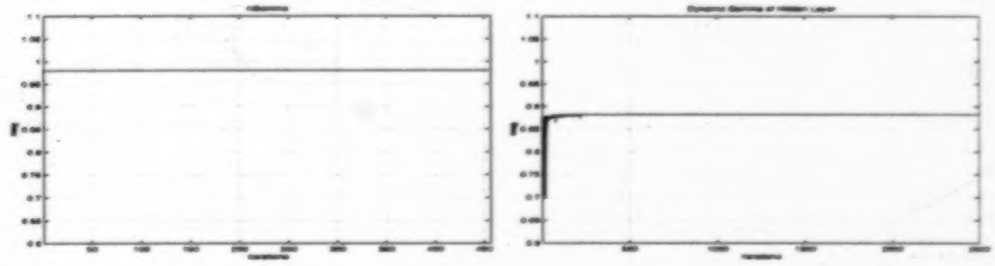
Figure 116: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

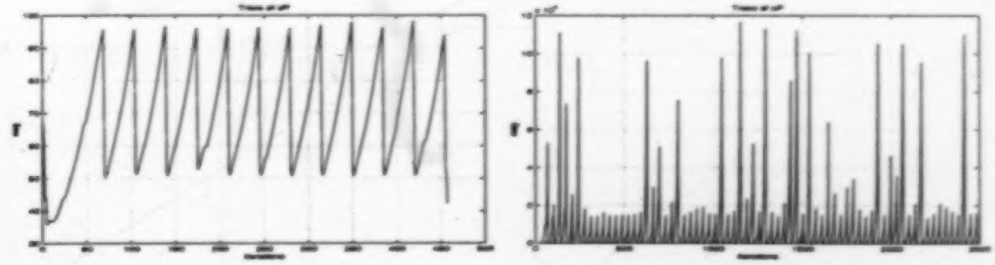
Figure 117: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

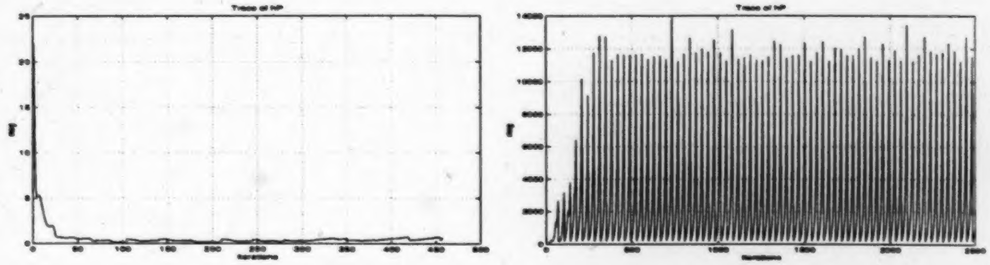
Figure 118: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

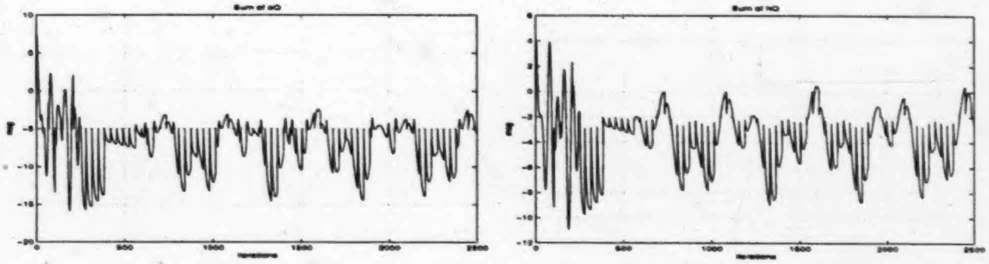
Figure 119: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Poorly alert**)

(b) Periodic resetting with VFF

Figure 120: Trace of covariance matrices in hidden layer

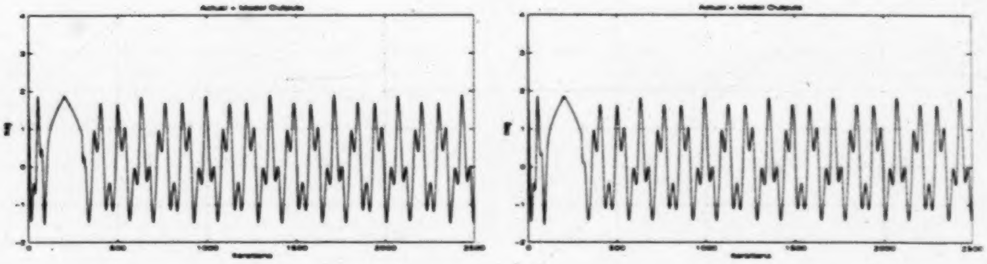


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 121: Sum of Q Elements

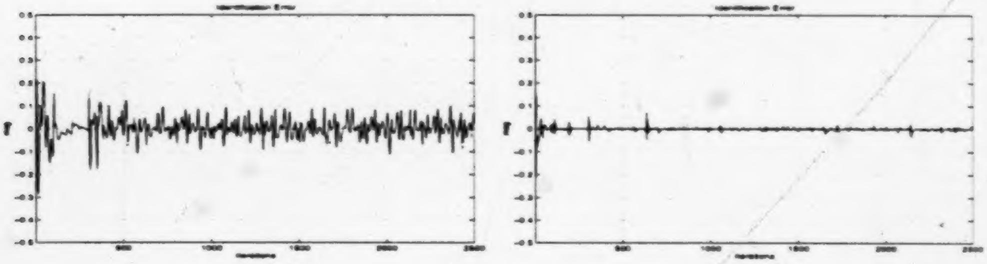
6.7.9 With Initial Weight Set 1 On Plant 3



(a) By conventional method

(b) By proposed method

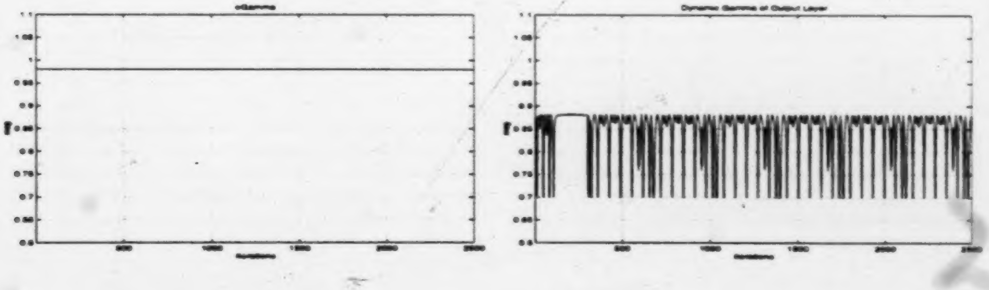
Figure 122: Actual output **identification** by model output



(a) By conventional method

(b) By proposed method

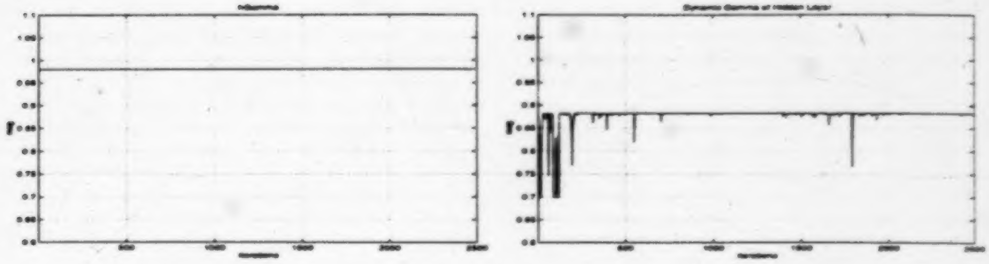
Figure 123: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

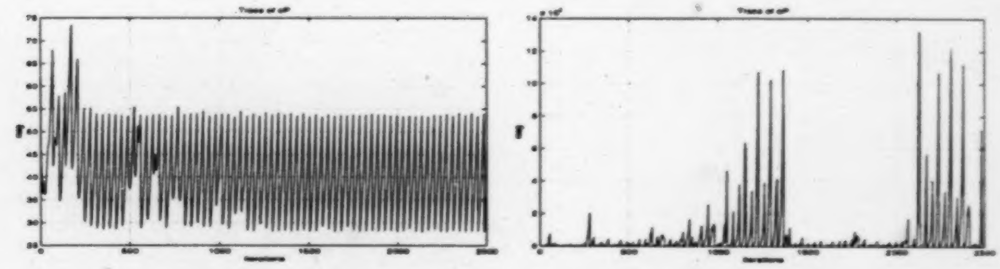
Figure 124: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

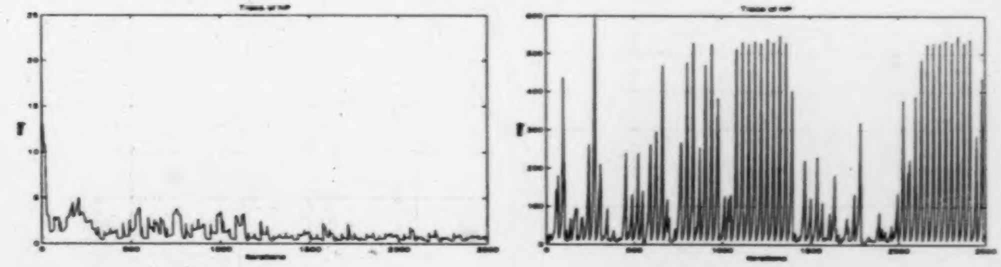
Figure 125: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

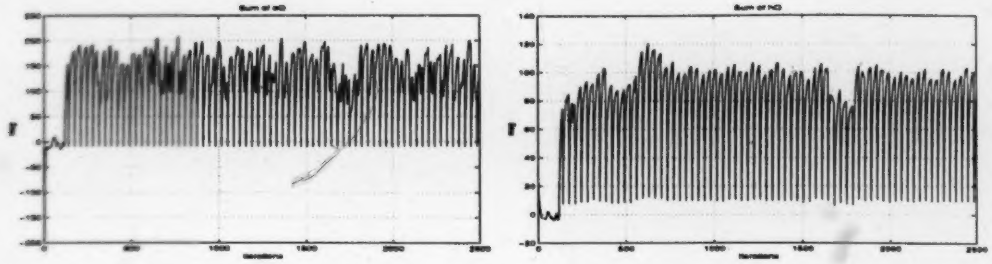
Figure 126: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

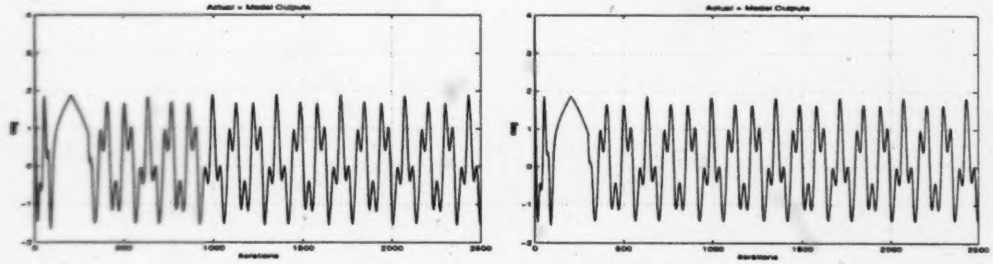
Figure 127: Trace of covariance matrices in hidden layer



(a) For proposed method in output layer (b) For proposed method in hidden layer

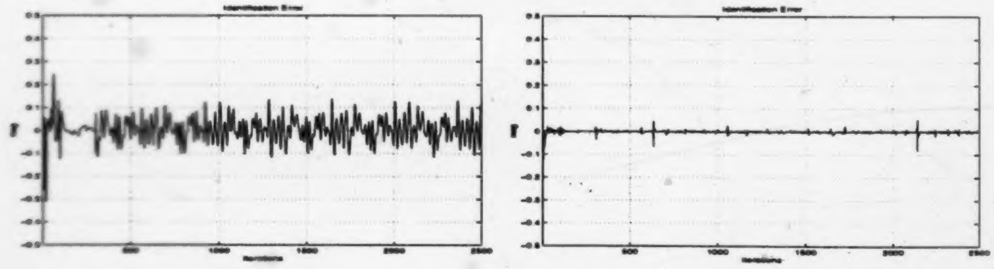
Figure 128: Sum of Q Elements

6.7.10 With Initial Weight Set 2 On Plant 3



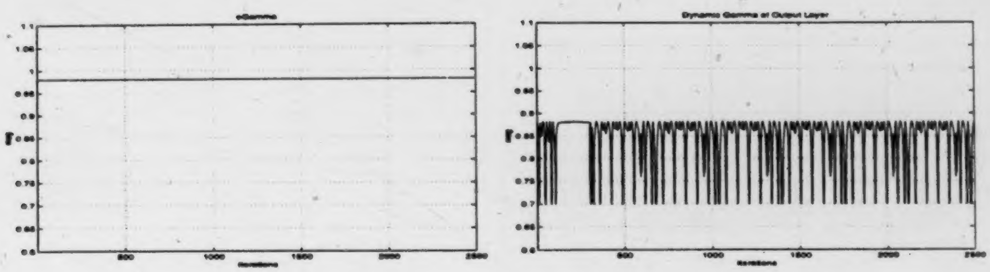
(a) By conventional method (b) By proposed method

Figure 129: Actual output **identification** by model output



(a) By conventional method (b) By proposed method

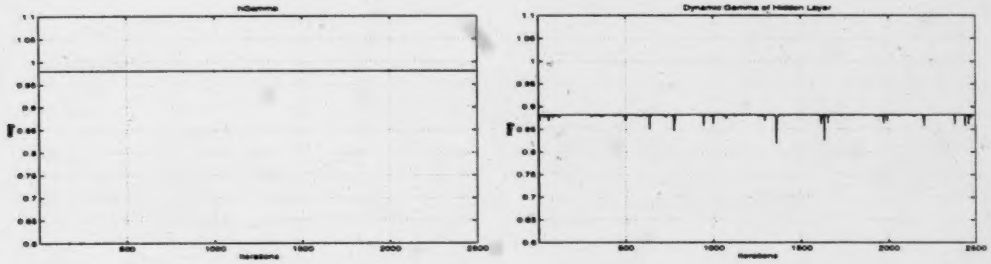
Figure 130: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

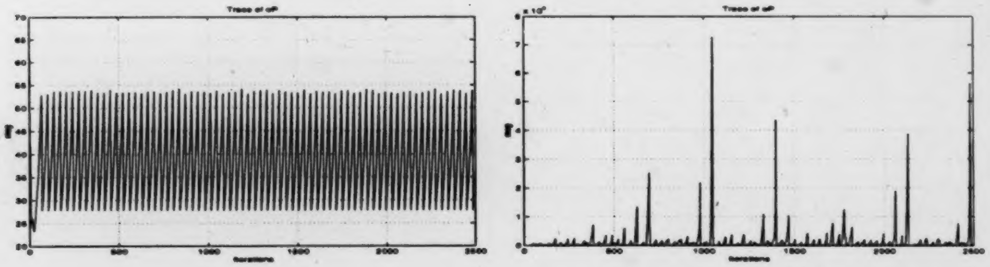
Figure 131: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

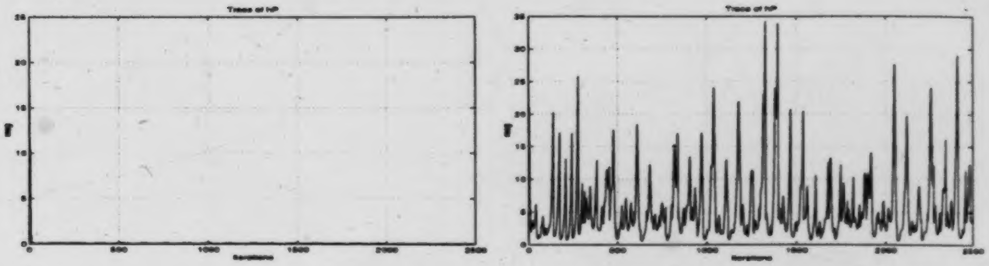
Figure 132: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

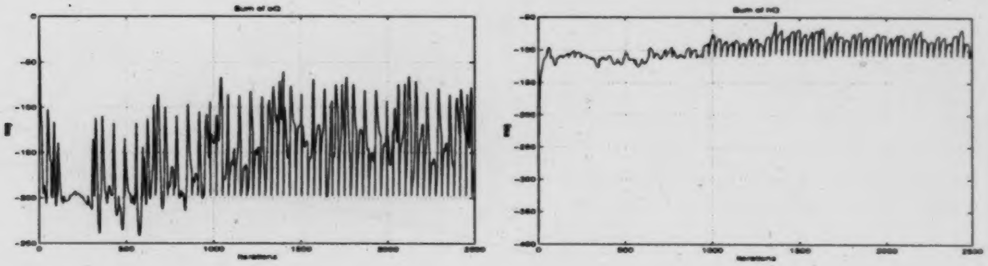
Figure 133: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (Poorly alert)

(b) Periodic resetting with VFF

Figure 134: Trace of covariance matrices in hidden layer

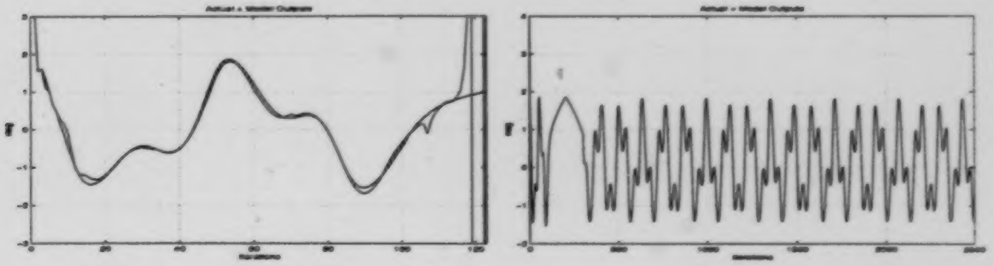


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 135: Sum of Q Elements

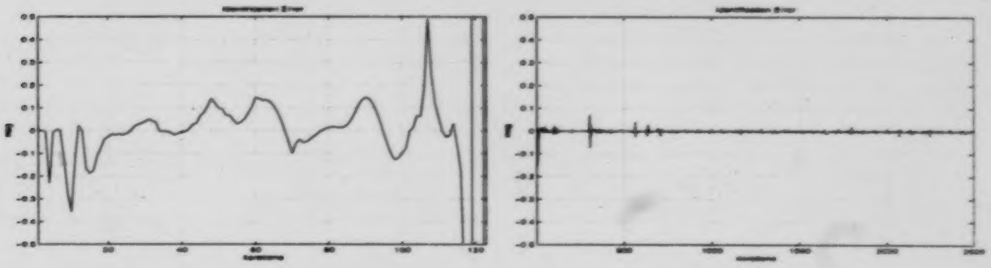
6.7.11 With Initial Weight Set 3 On Plant 3



(a) By conventional method (**Wind-up**)

(b) By proposed method

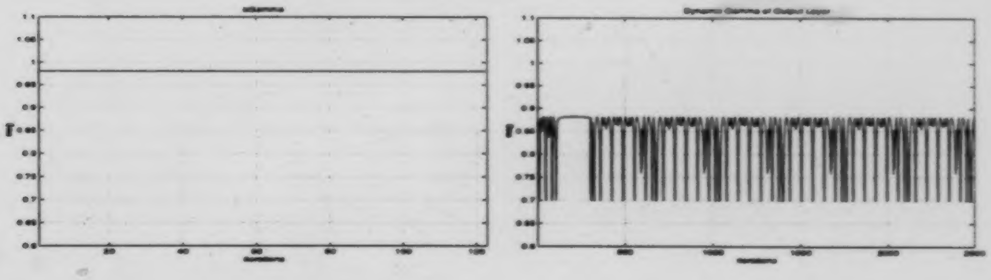
Figure 136: Actual output **identification** by model output



(a) By conventional method (**Burst**)

(b) By proposed method

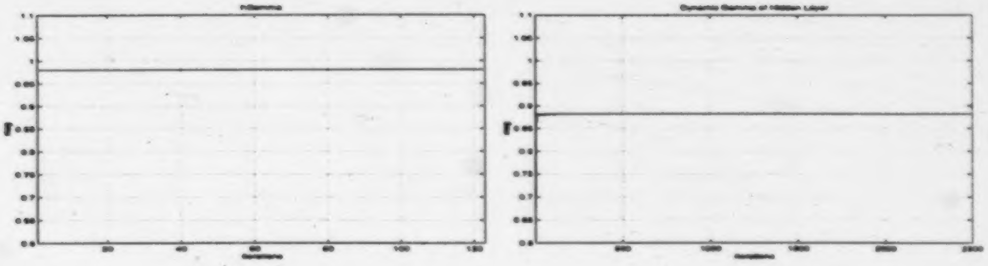
Figure 137: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

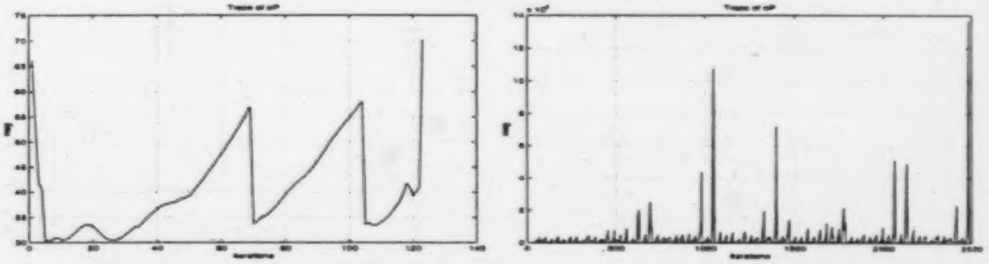
Figure 138: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

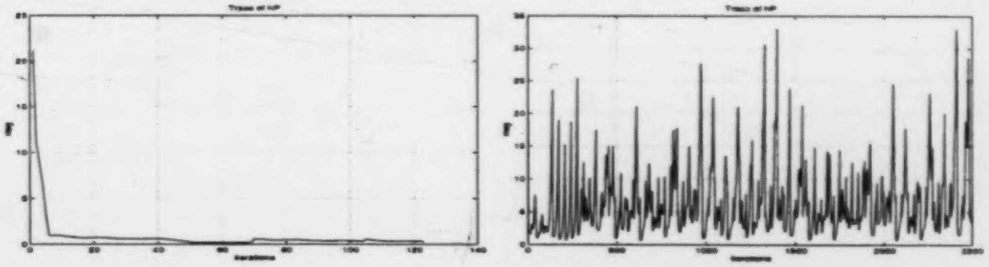
Figure 139: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

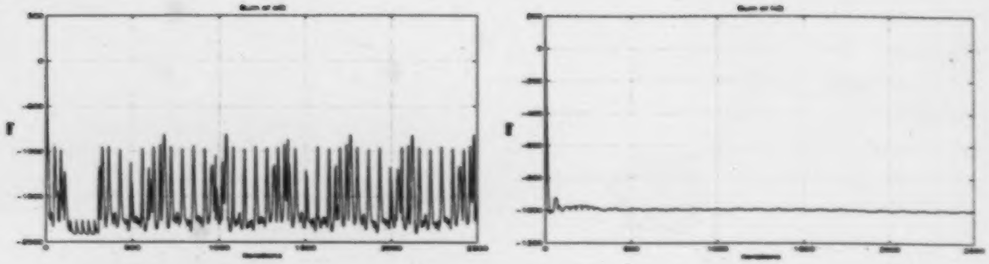
Figure 140: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (Poorly alert)

(b) Periodic resetting with VFF

Figure 141: Trace of covariance matrices in hidden layer

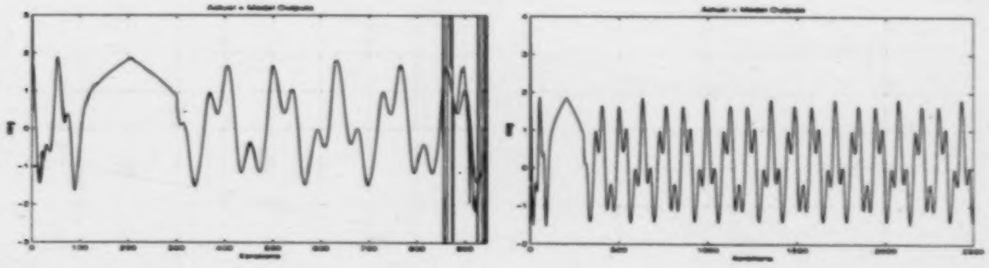


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 142: Sum of Q Elements

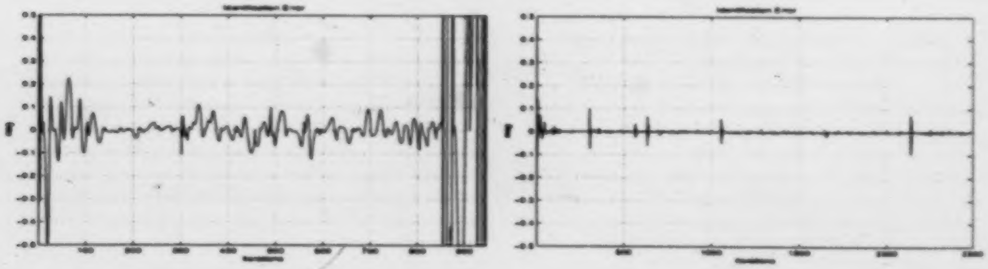
6.7.12 With Initial Weight Set 4 (Random Number) On Plant 3



(a) By conventional method (**Wind-up**)

(b) By proposed method

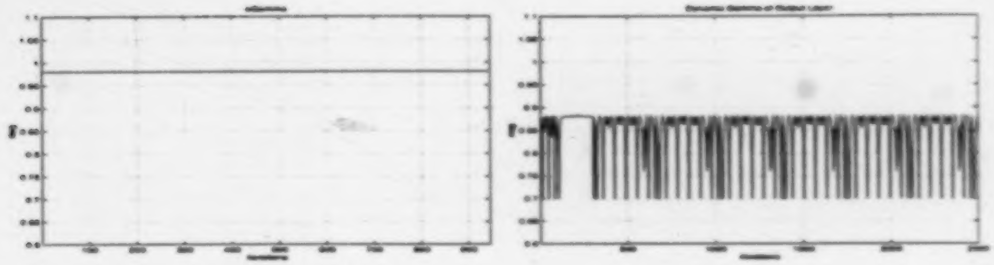
Figure 143: Actual output **identification** by model output



(a) By conventional method (**Burst**)

(b) By proposed method

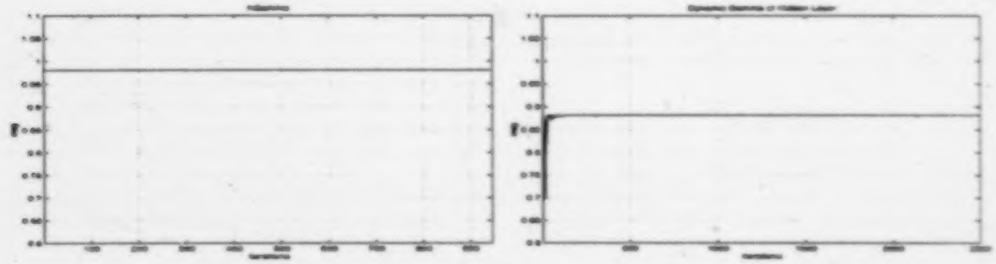
Figure 144: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

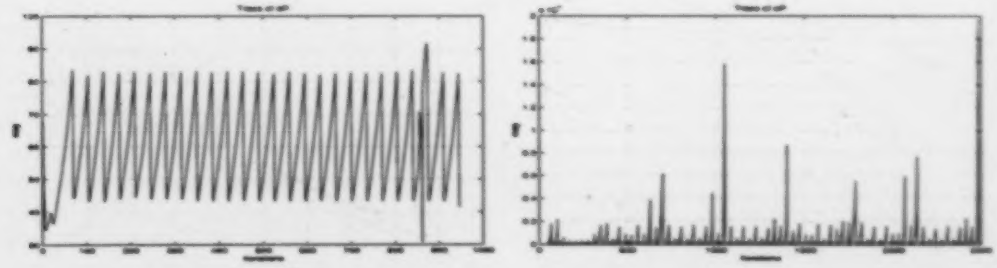
Figure 145: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

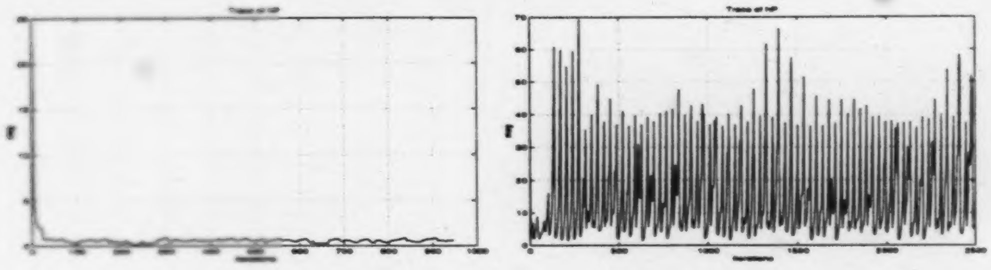
Figure 146: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

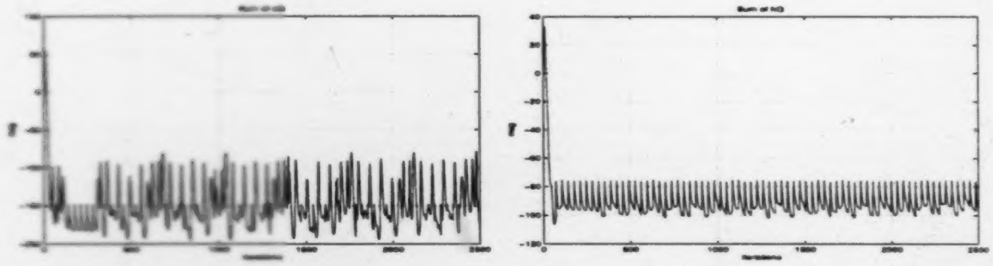
Figure 147: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (Poor frequency)

(b) Periodic resetting with VFF

Figure 148: Trace of covariance matrices in hidden layer



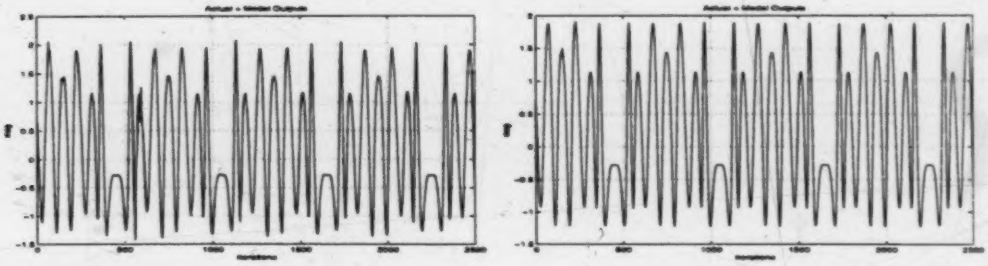
(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 149: Sum of Q Elements

With regard to simulation results of plant 3 with initial weights set 4 (random number), a clear clue was not found for the conventional method leading to *wind-up*, because at least, the trace of the covariance matrices in the output and hidden layers looks *alert* as shown in (a) Figure in 147 and 148. One possible interpretation for the burst phenomenon by the conventional method is its lack of capability to recover the lost information occurring from periodic resetting of the covariance matrix. Another explanation can be made from having violated the fact that the input for persistent excitation must have sufficiently *rich frequency* content. However, Figure 148 (a) does not show rich frequency.

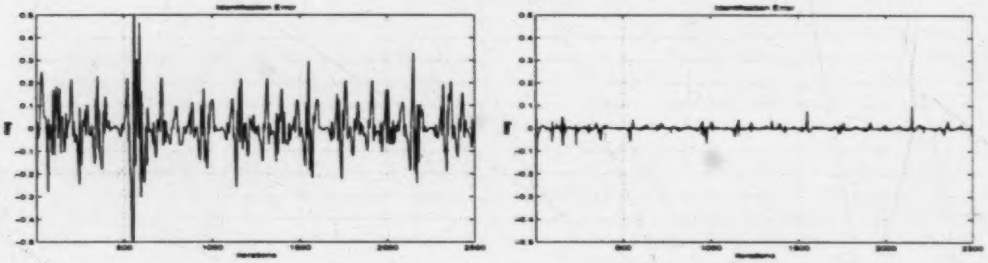
6.7.13 With Initial Weight Set 1 On Plant 4



(a) By conventional method

(b) By proposed method

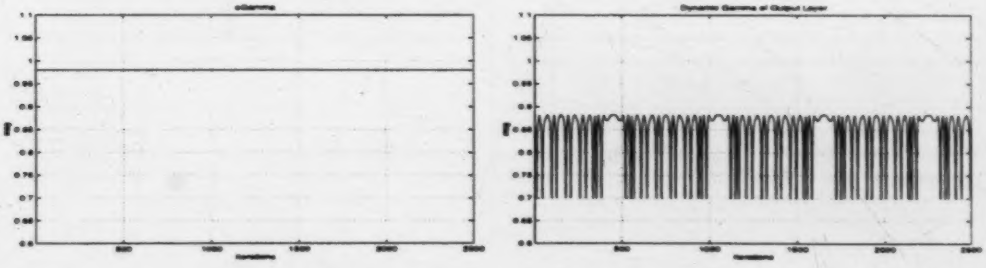
Figure 150: Actual output **identification** by model output



(a) By conventional method (**Poor**)

(b) By proposed method

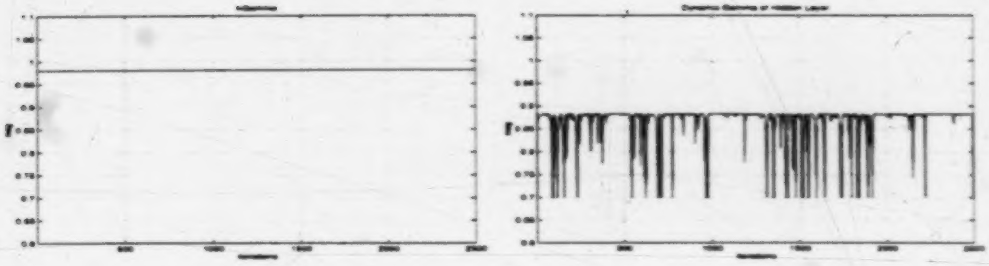
Figure 151: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

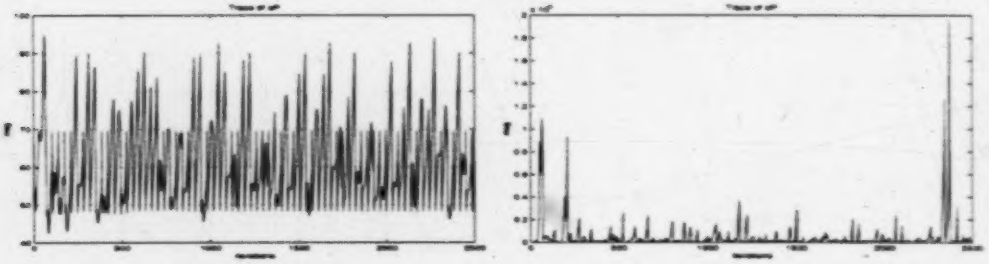
Figure 152: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

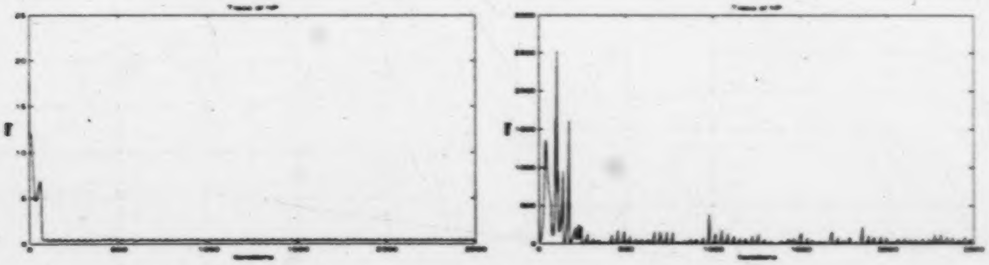
Figure 153: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

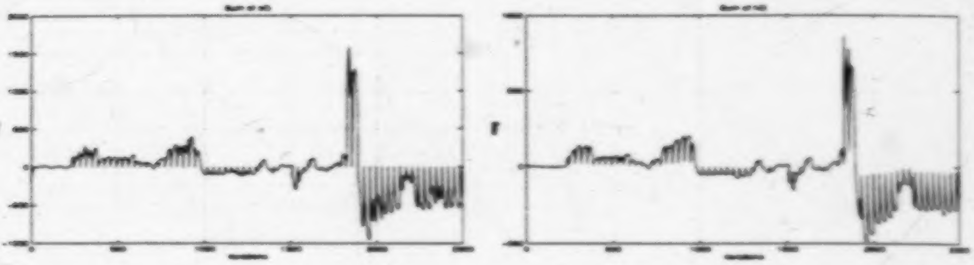
Figure 154: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Poorly alert**)

(b) Periodic resetting with VFF

Figure 155: Trace of covariance matrices in hidden layer

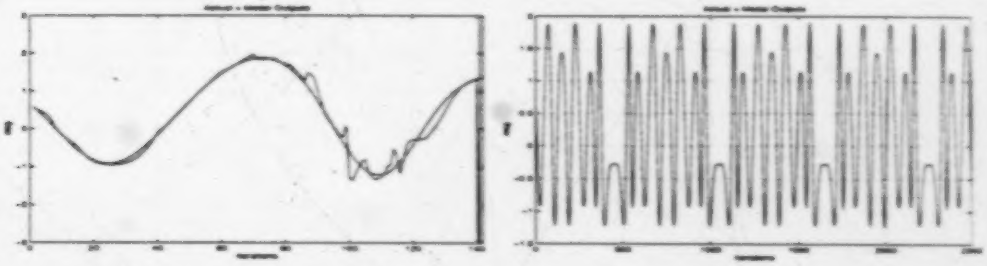


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 156: Sum of Q Elements

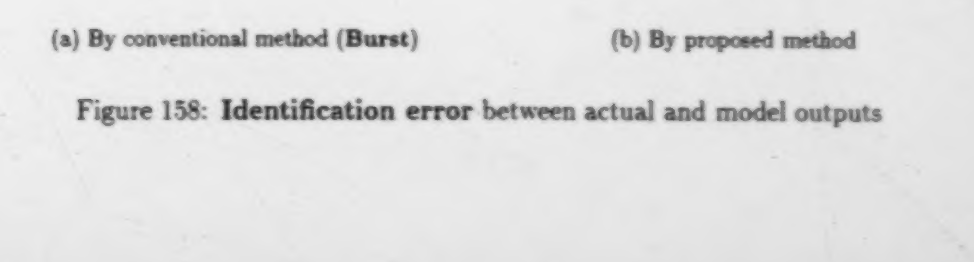
6.7.14 With Initial Weight Set 2 On Plant 4



(a) By conventional method (**Wind-up**)

(b) By proposed method

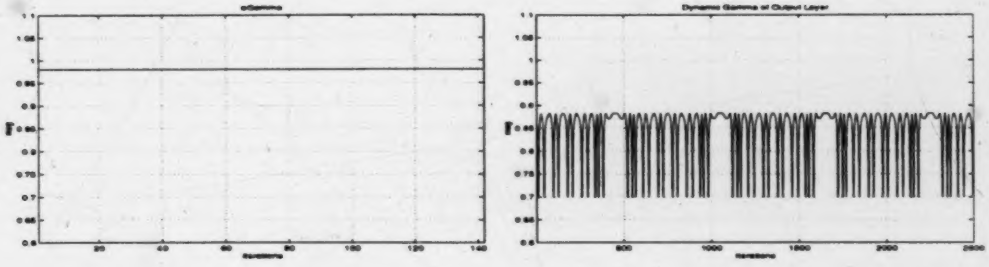
Figure 157: Actual output **identification** by model output



(a) By conventional method (**Burst**)

(b) By proposed method

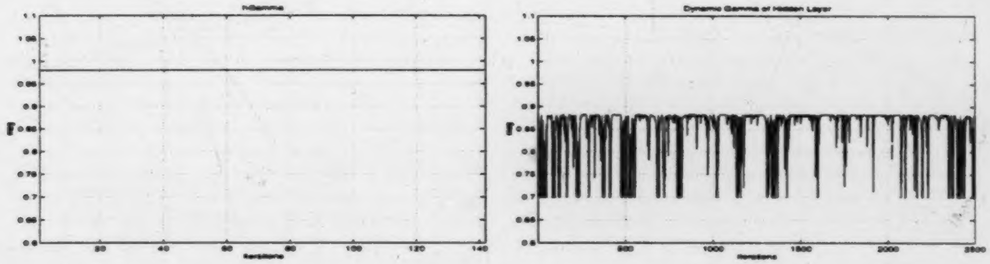
Figure 158: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

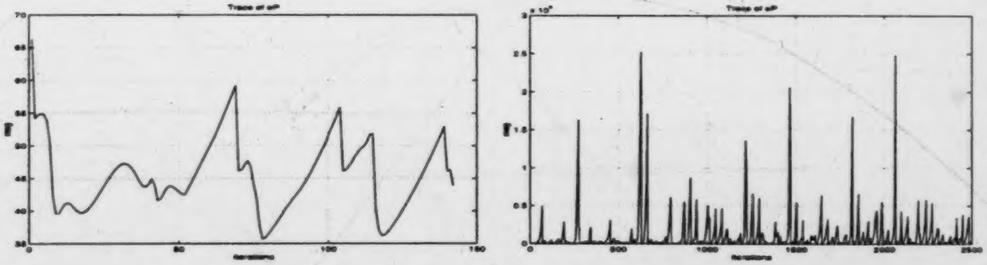
Figure 159: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

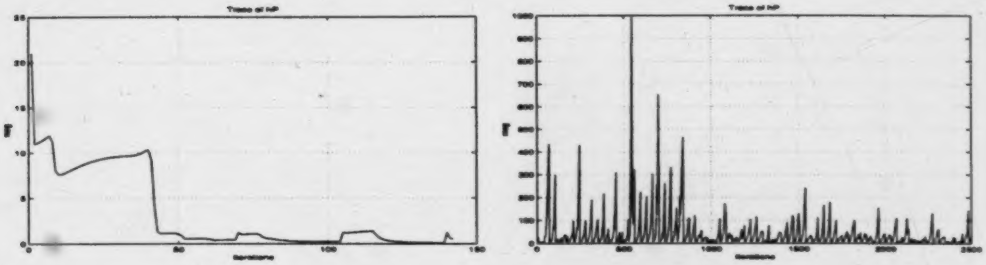
Figure 160: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

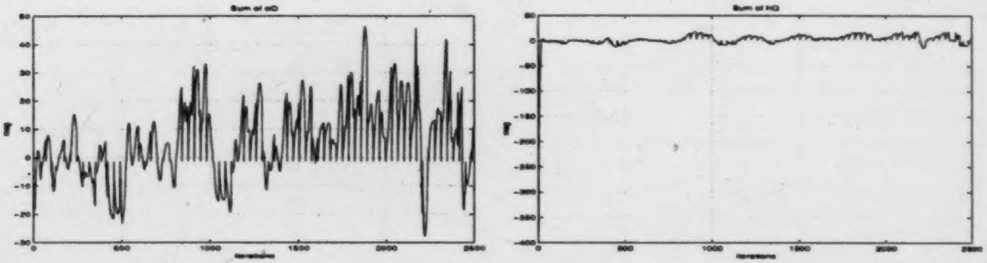
Figure 161: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Poorly alert**)

(b) Periodic resetting with VFF

Figure 162: Trace of covariance matrices in hidden layer

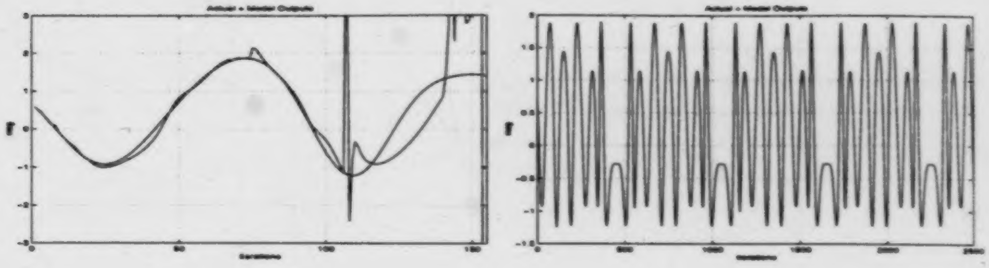


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 163: Sum of Q Elements

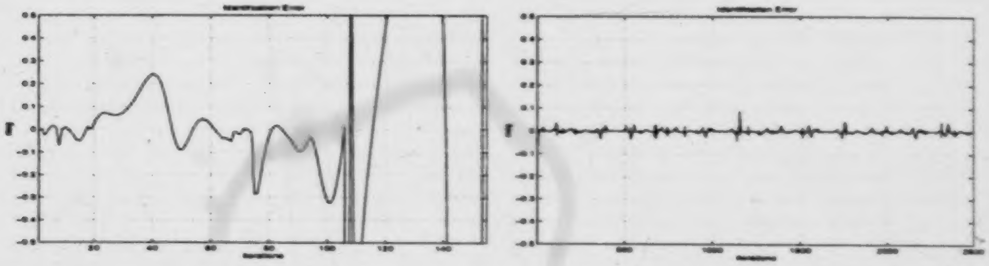
6.7.15 With Initial Weight Set 3 On Plant 4



(a) By conventional method (**Wind-up**)

(b) By proposed method

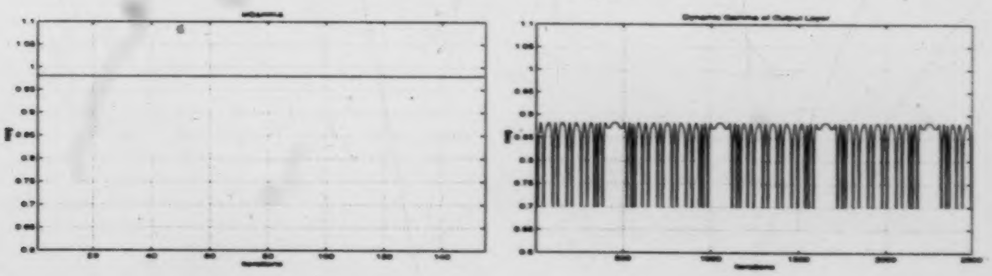
Figure 164: Actual output **identification** by model output



(a) By conventional method (**Burst**)

(b) By proposed method

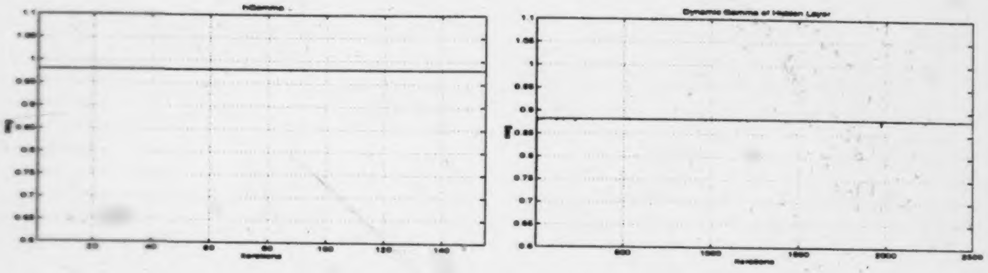
Figure 165: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

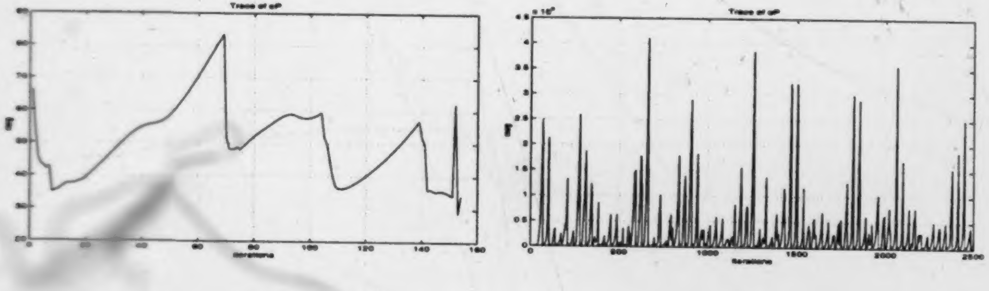
Figure 166: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

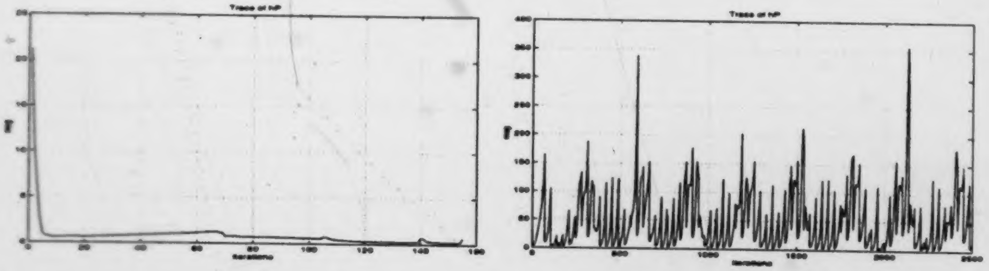
Figure 167: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

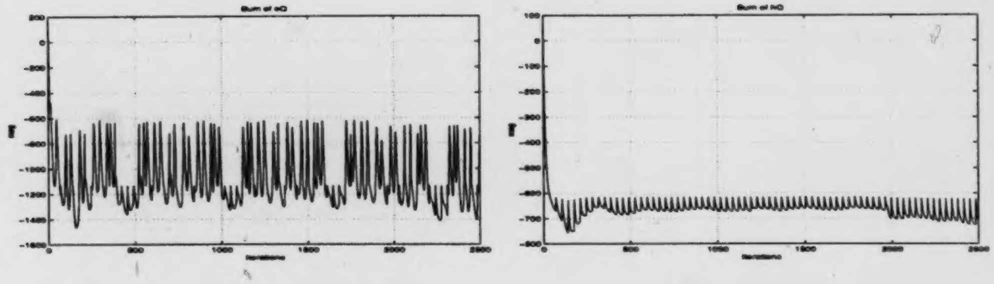
Figure 168: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF **(Poorly alert)**

(b) Periodic resetting with VFF

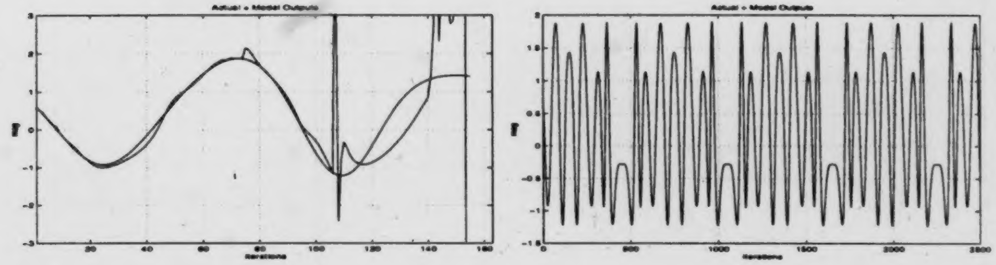
Figure 169: Trace of covariance matrices in hidden layer



(a) For proposed method in output layer (b) For proposed method in hidden layer

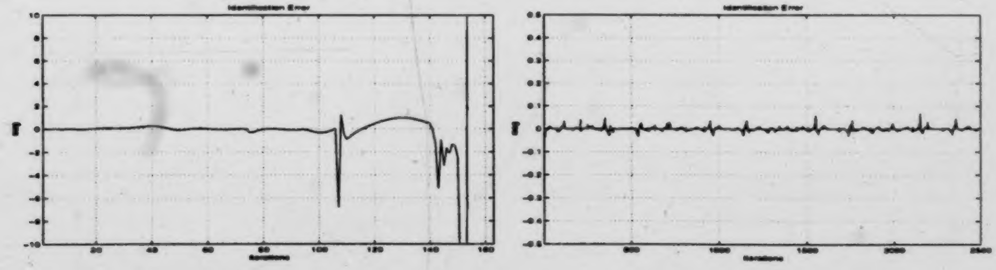
Figure 170: Sum of Q Elements

6.7.16 With Initial Weight Set 4 (Random Number) On Plant 4



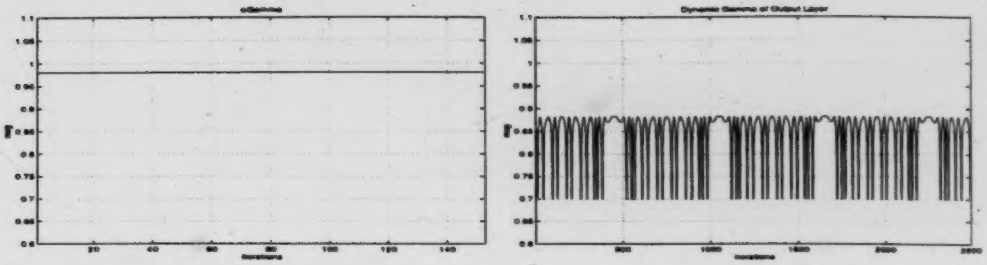
(a) By conventional method (**Wind-up**) (b) By proposed method

Figure 171: Actual output **identification** by model output



(a) By conventional method (**Burst**) (b) By proposed method

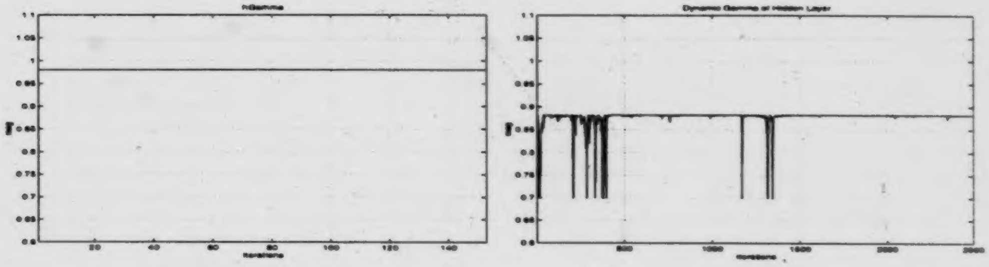
Figure 172: **Identification error** between actual and model outputs



(a) Conventional CFF= 0.98

(b) Proposed VFF

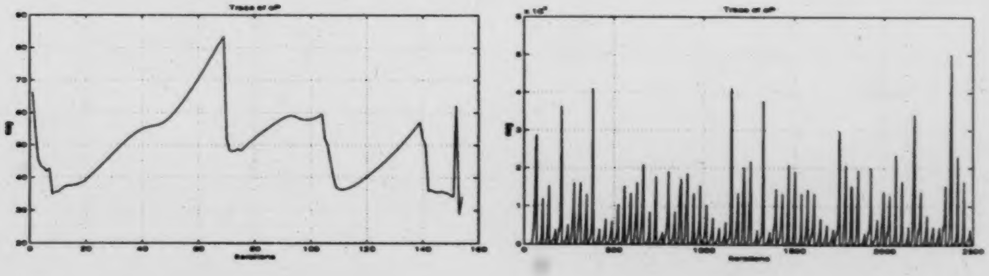
Figure 173: Constant and Variable Forgetting Factors in output layer



(a) Conventional CFF= 0.98

(b) Proposed VFF

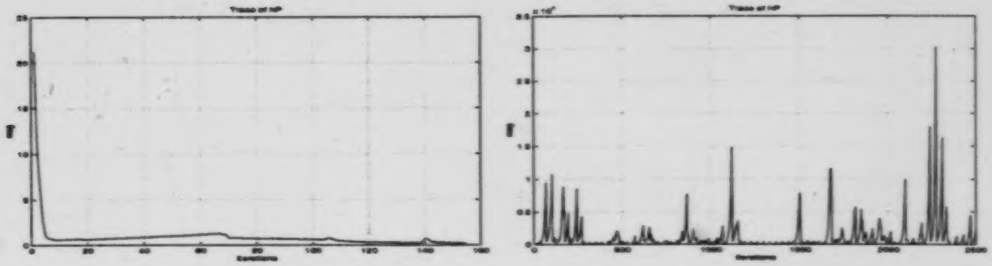
Figure 174: Constant and Variable Forgetting Factors in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

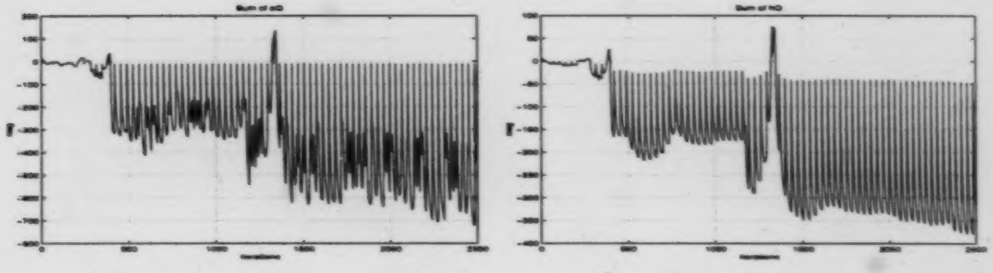
Figure 175: Trace of covariance matrices in output layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

Figure 176: Trace of covariance matrices in hidden layer

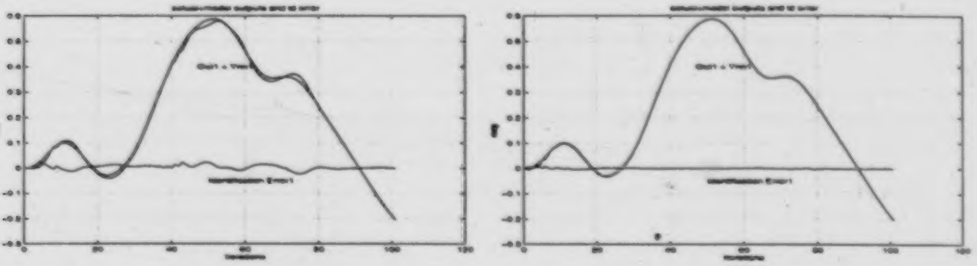


(a) For proposed method in output layer

(b) For proposed method in hidden layer

Figure 177: Sum of Q Elements

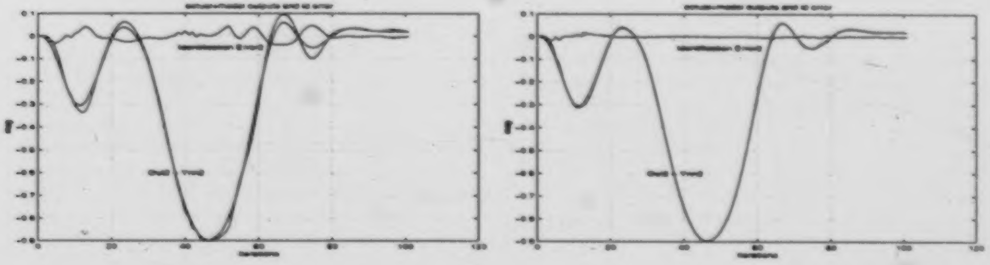
6.7.17 With Initial Weight Set 1 On MIMO Plant 5 (Swing Leg)



(a) By conventional method

(b) By proposed method

Figure 178: Actual output1 **identification** and **error** by model output1

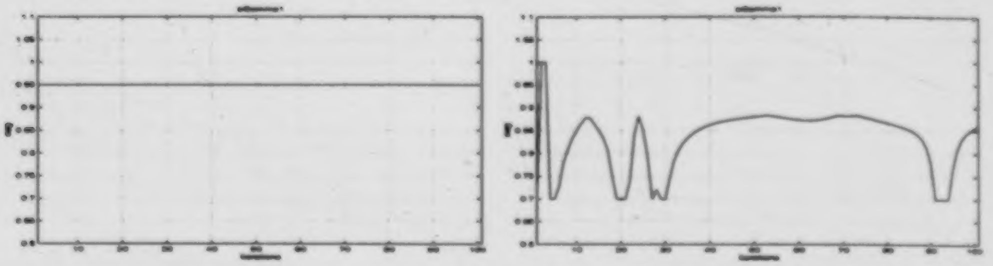


(a) By conventional method (**Poor Id. error**)

(b) By proposed method

Figure 179: Actual output2 **identification** and **error** by model output2

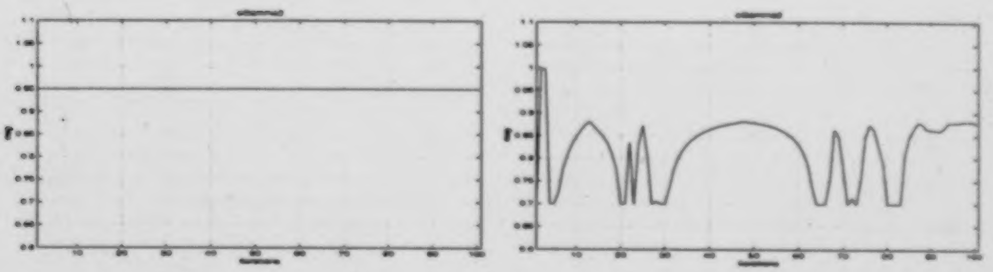
The proposed method is better than the conventional method in terms of the identification error with initial weight set 1.



(a) Conventional CFF= 0.95

(b) Proposed VFF

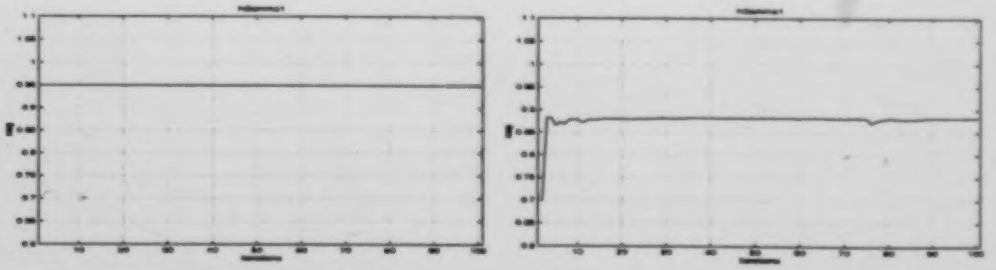
Figure 180: Constant and Variable Forgetting Factors at output1 in output layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

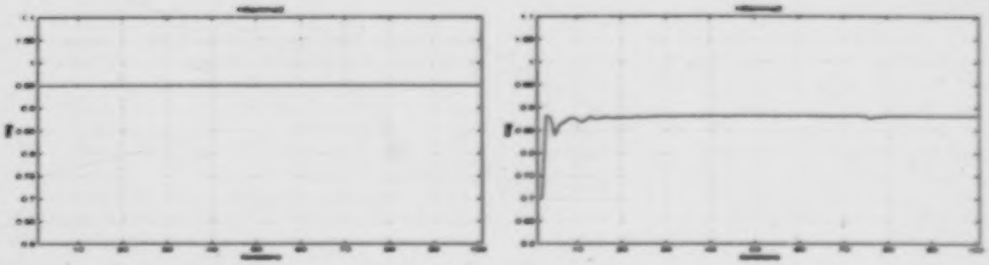
Figure 181: Constant and Variable Forgetting Factors at output2 in output layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

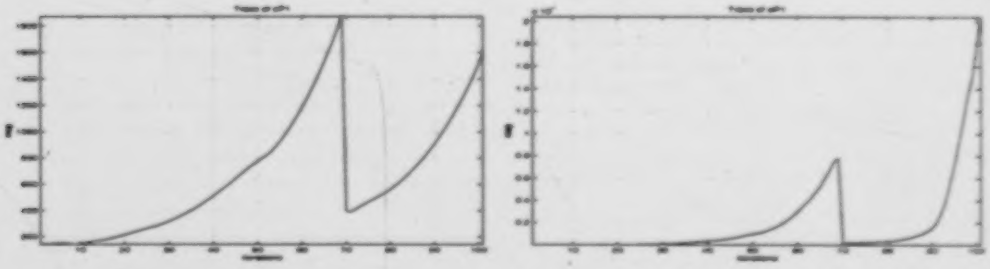
Figure 182: Constant and Variable Forgetting Factors at output1 in hidden layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

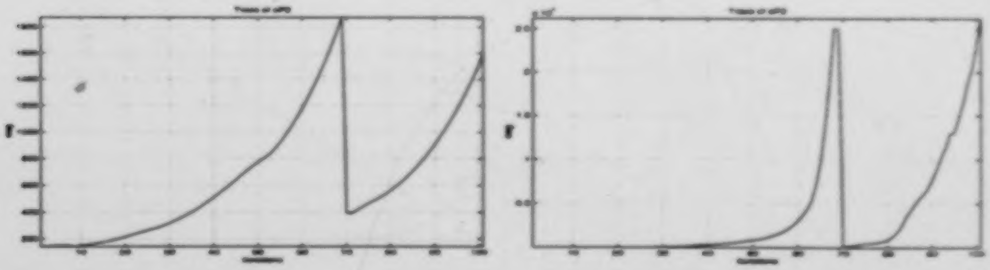
Figure 183: Constant and Variable Forgetting Factors at output2 in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

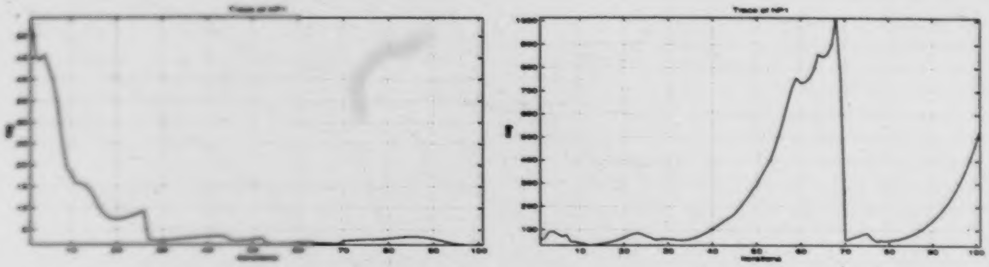
Figure 184: Trace of covariance matrices at output1 in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

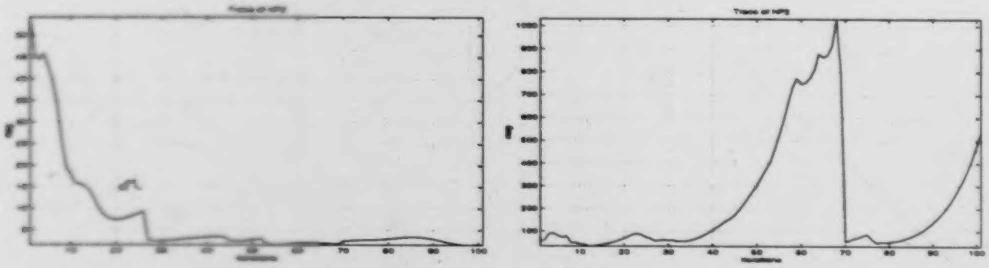
Figure 185: Trace of covariance matrices at output2 in output layer



(a) Periodic resetting with CFF (**Poorly alert**)

(b) Periodic resetting with VFF

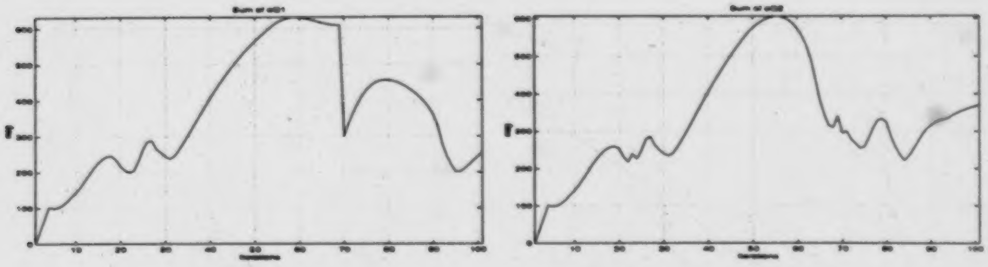
Figure 186: Trace of covariance matrices at output1 in hidden layer



(a) Periodic resetting with CFF (**Poorly alert**)

(b) Periodic resetting with VFF

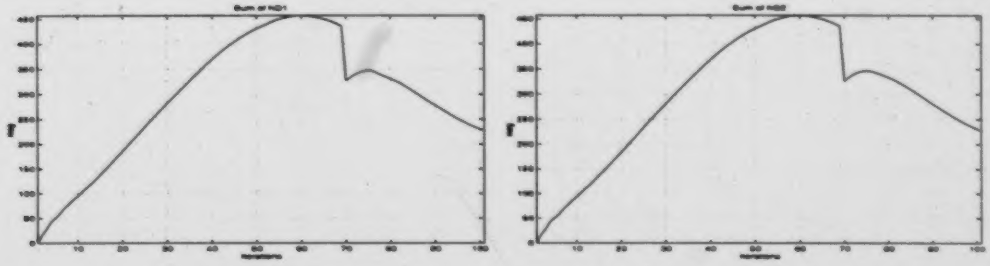
Figure 187: Trace of covariance matrices at output2 in hidden layer



(a) At Output 1

(b) At Output 2

Figure 188: Sum of Q elements (proposed) in *output layer*

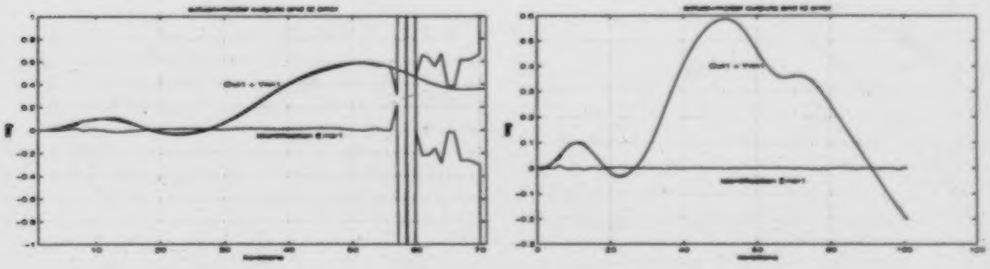


(a) At Output 1

(b) At Output 2

Figure 189: Sum of Q (proposed) elements in *hidden layer*

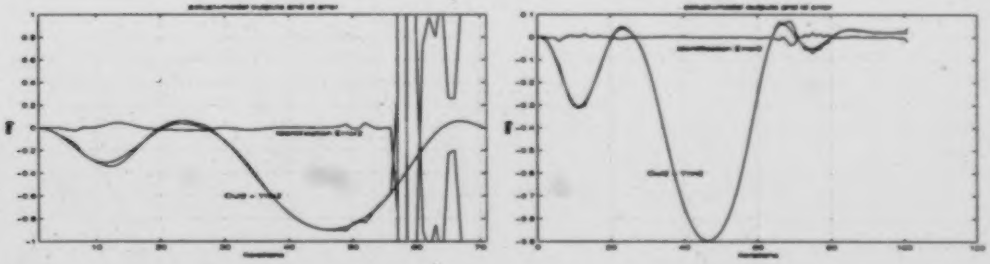
6.7.18 With Initial Weight Set 2 On MIMO Plant 5 (Swing Leg)



(a) By conventional method (**Wind-up**)

(b) By proposed method

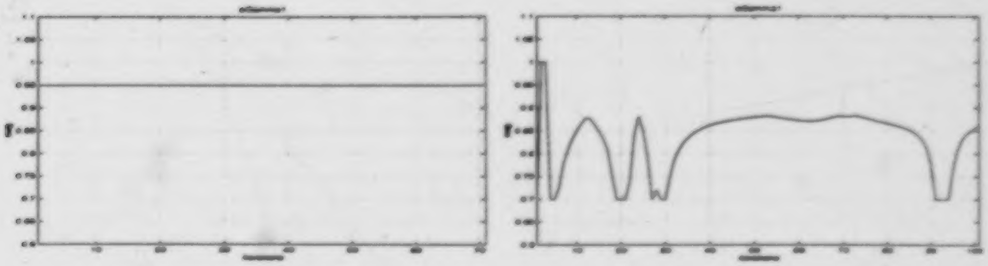
Figure 190: Actual output1 **identification** and **error** by model output1



(a) By conventional method (**Wind-up**)

(b) By proposed method

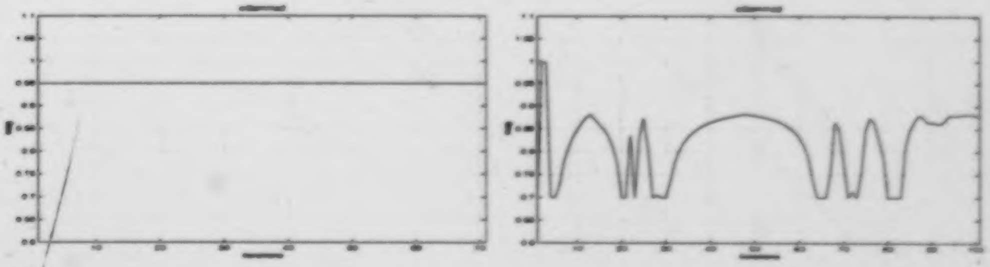
Figure 191: Actual output2 **identification** and **error** by model output2



(a) Conventional CFF= 0.95

(b) Proposed VFF

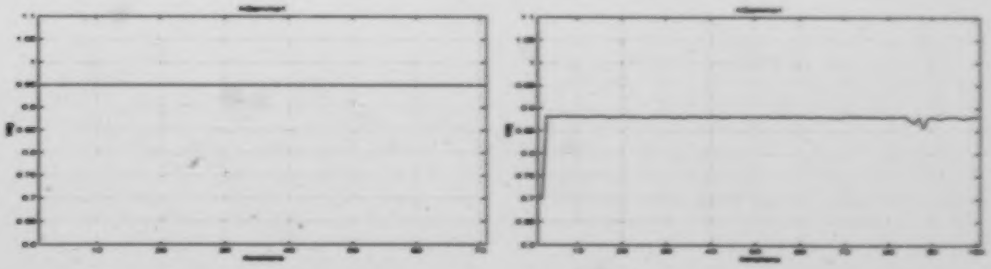
Figure 192: Constant and Variable Forgetting Factors at output1 in output layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

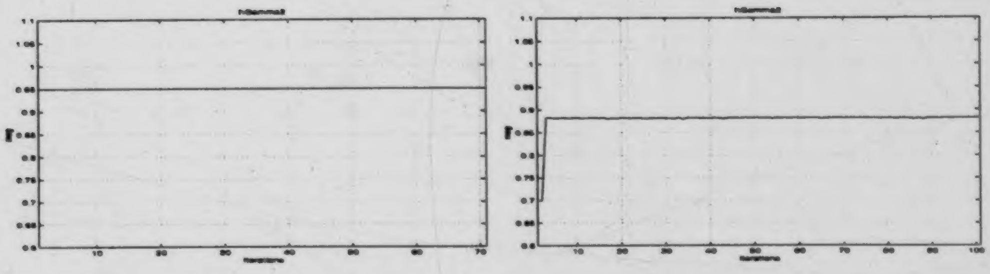
Figure 193: Constant and Variable Forgetting Factors at output2 in output layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

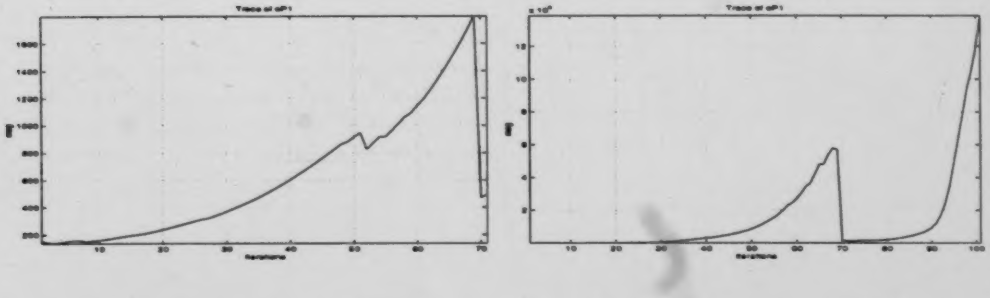
Figure 194: Constant and Variable Forgetting Factors at output1 in hidden layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

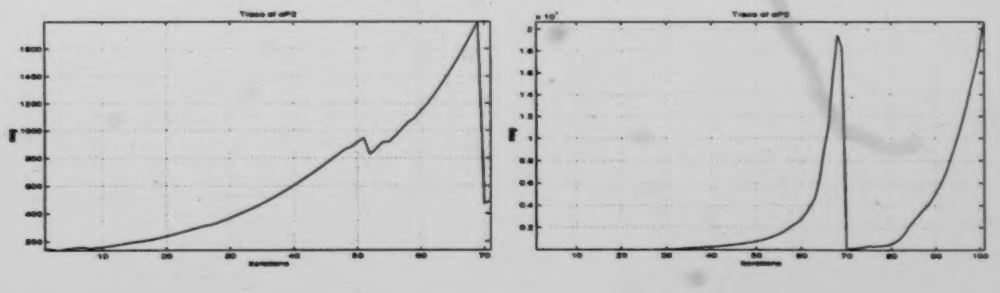
Figure 195: Constant and Variable Forgetting Factors at output2 in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

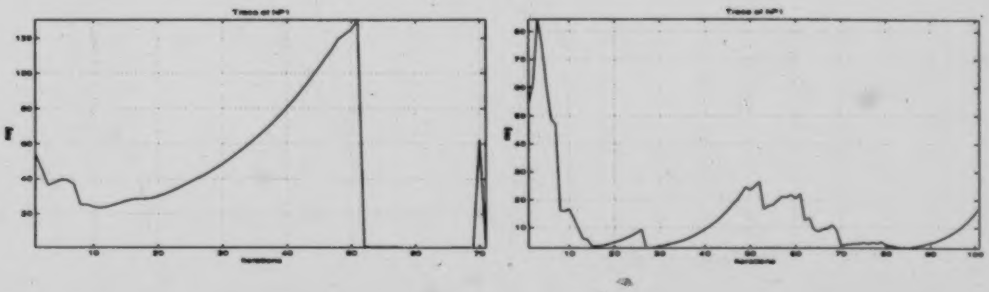
Figure 196: Trace of covariance matrices at output1 in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

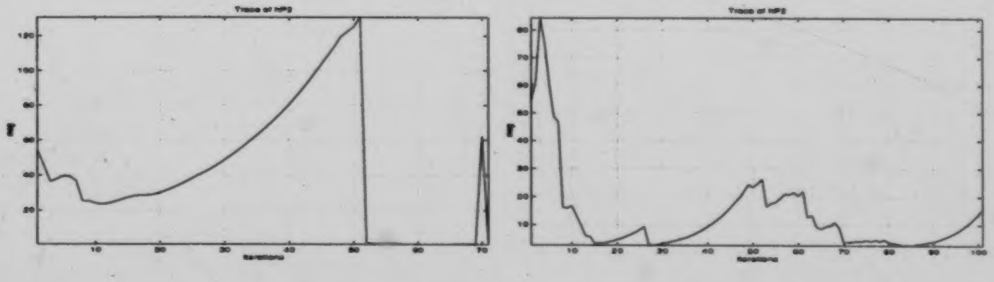
Figure 197: Trace of covariance matrices at output2 in output layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

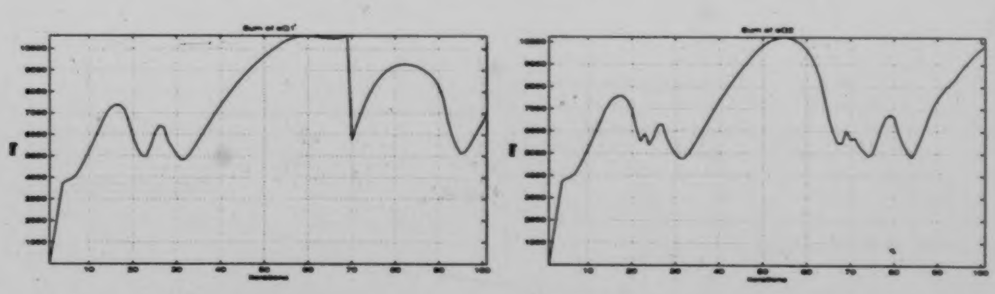
Figure 198: Trace of covariance matrices at output1 in hidden layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

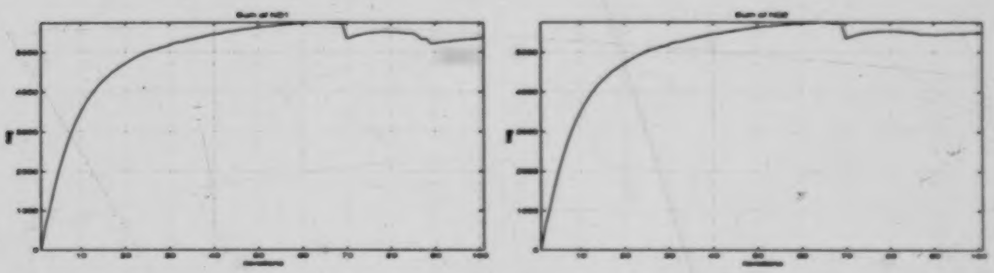
Figure 199: Trace of covariance matrices at output2 in hidden layer



(a) At Output 1

(b) At Output 2

Figure 200: Sum of Q elements (proposed) in *output layer*

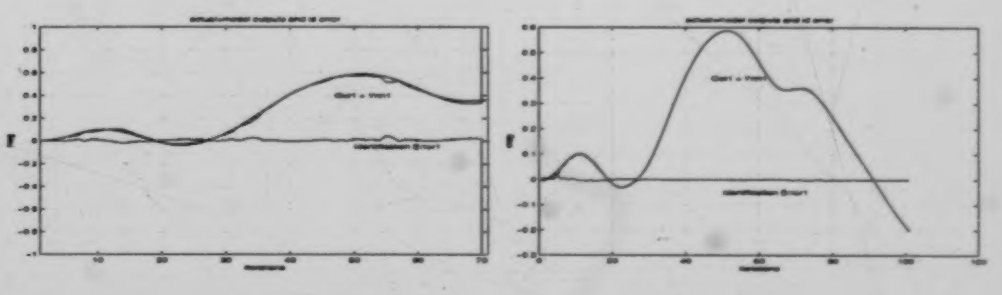


(a) At Output 1

(b) At Output 2

Figure 201: Sum of Q elements (proposed) in *hidden layer*

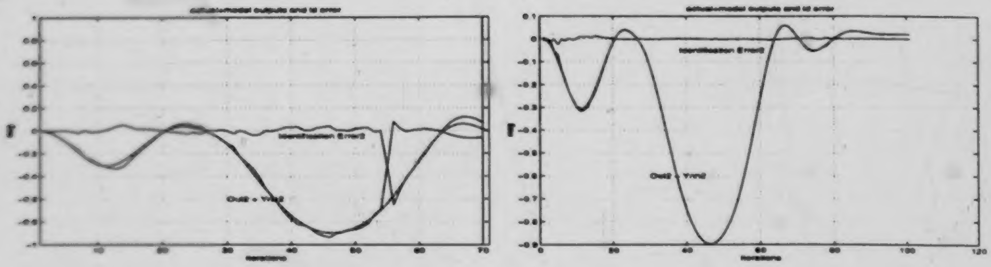
6.7.19 With Initial Weight Set 3 On MIMO Plant 5 (Swing Leg)



(a) By conventional method (**Wind-up**)

(b) By proposed method

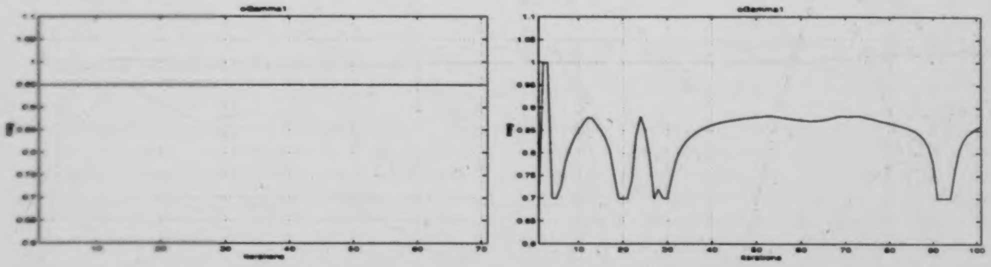
Figure 202: Actual output1 **identification** and **error** by model output1



(a) By conventional method (**Wind-up**)

(b) By proposed method

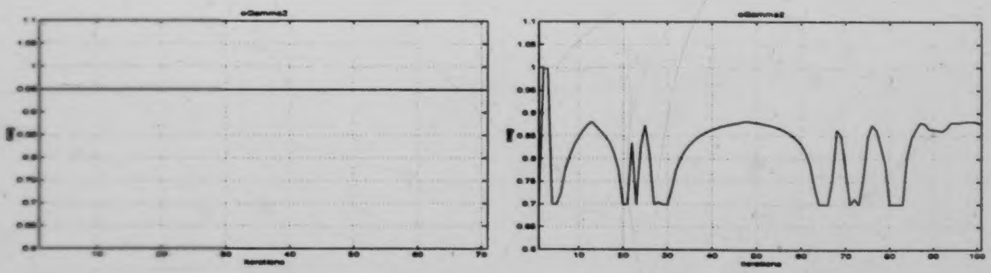
Figure 203: Actual output2 **identification** and **error** by model output2



(a) Conventional CFF= 0.95

(b) Proposed VFF

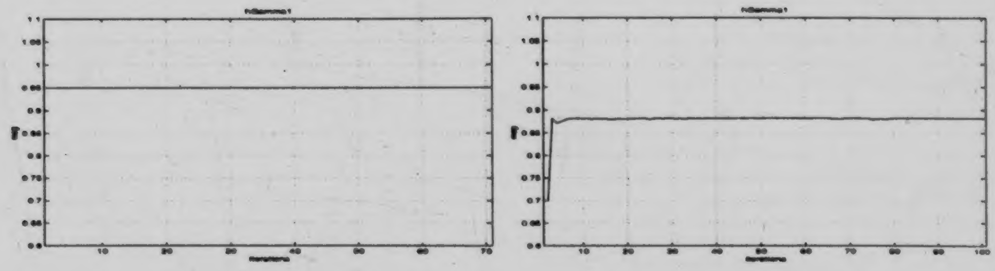
Figure 204: Constant and Variable Forgetting Factors at output1 in output layer



(a) Conventional CFF= 0.95

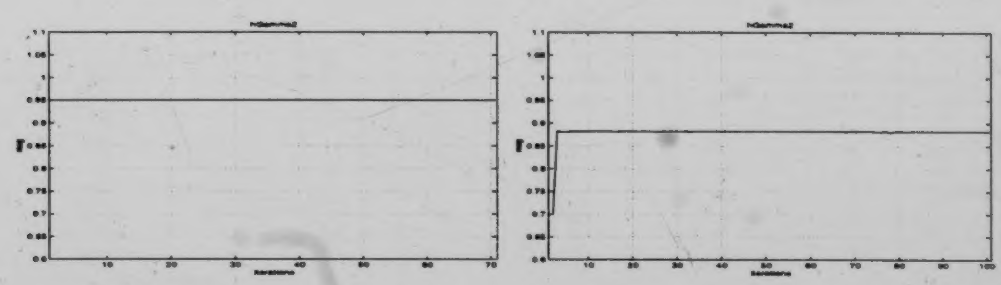
(b) Proposed VFF

Figure 205: Constant and Variable Forgetting Factors at output2 in output layer



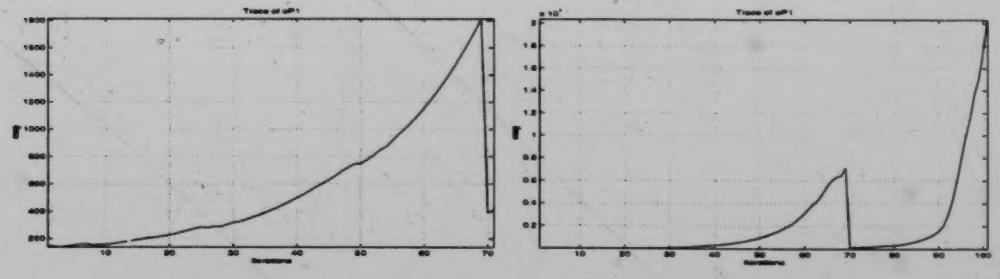
(a) Conventional CFF= 0.95 (b) Proposed VFF

Figure 206: Constant and Variable Forgetting Factors at output1 in hidden layer



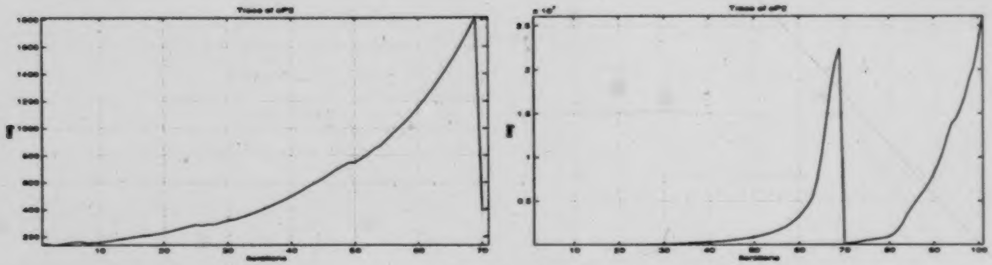
(a) Conventional CFF= 0.95 (b) Proposed VFF

Figure 207: Constant and Variable Forgetting Factors at output2 in hidden layer



(a) Periodic resetting with CFF (b) Periodic resetting with VFF

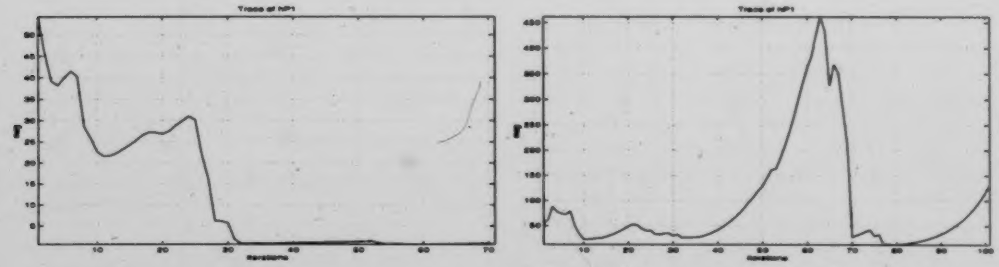
Figure 208: Trace of covariance matrices at output1 in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

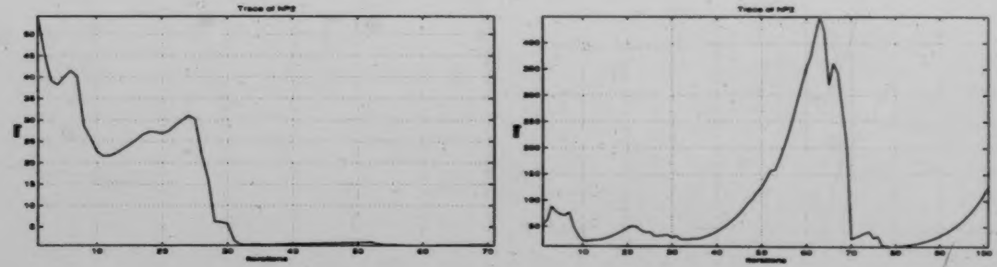
Figure 209: Trace of covariance matrices at output2 in output layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

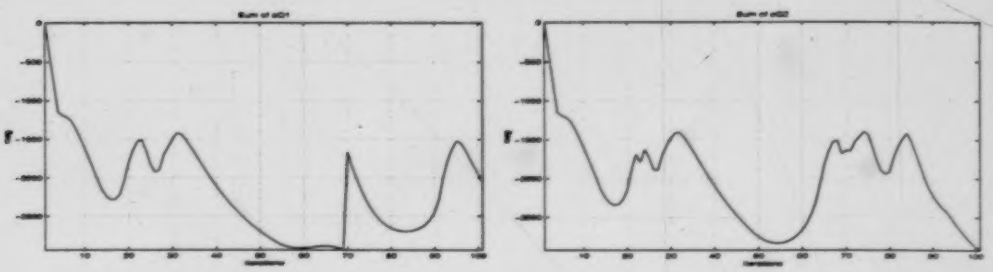
Figure 210: Trace of covariance matrices at output1 in hidden layer



(a) Periodic resetting with CFF (**Not alert**)

(b) Periodic resetting with VFF

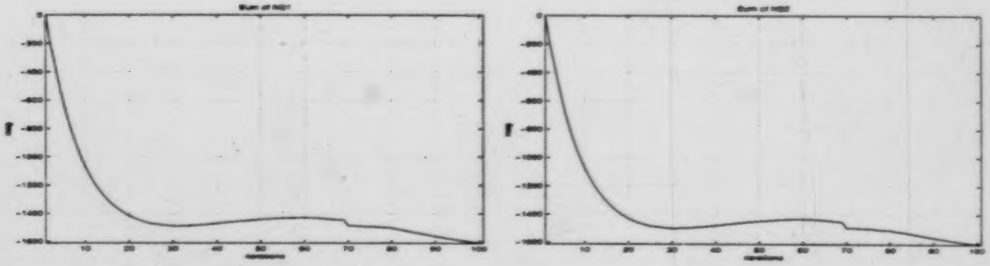
Figure 211: Trace of covariance matrices at output2 in hidden layer



(a) At Output 1

(b) At Output 2

Figure 212: Sum of Q elements (proposed) in *output layer*

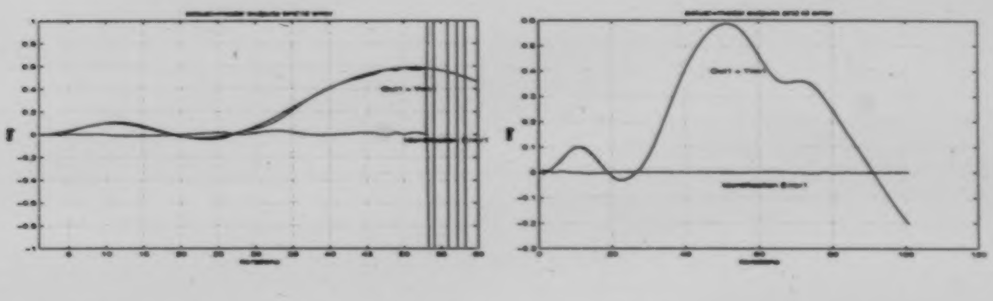


(a) At Output 1

(b) At Output 2

Figure 213: Sum of Q elements (proposed) in *hidden layer*

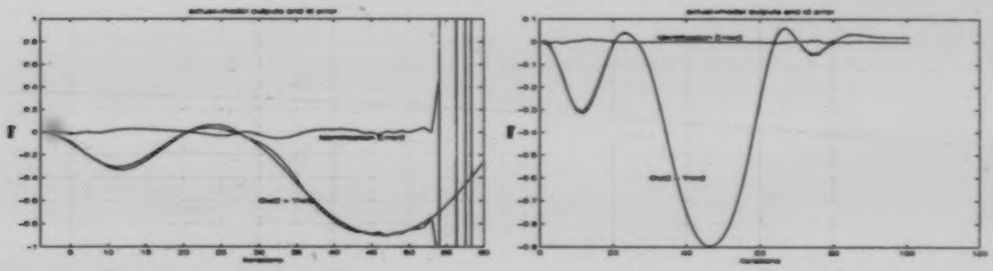
6.7.20 With Initial Weight Set4 (Random Number) On Plant 5 (Swing Leg)



(a) By conventional method (**Wind-up**)

(b) By proposed method

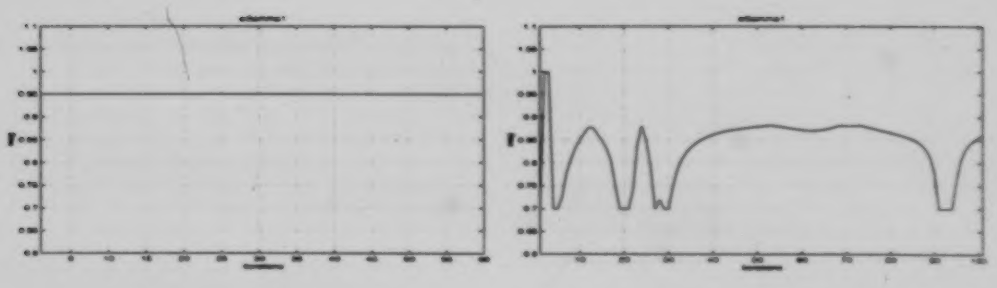
Figure 214: Actual output1 **identification** and **error** by model output1



(a) By conventional method (**Wind-up**)

(b) By proposed method

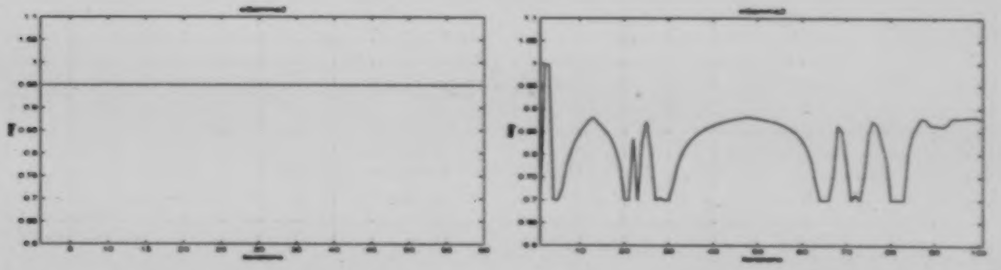
Figure 215: Actual output2 **identification** and **error** by model output2



(a) Conventional CFF= 0.95

(b) Proposed VFF

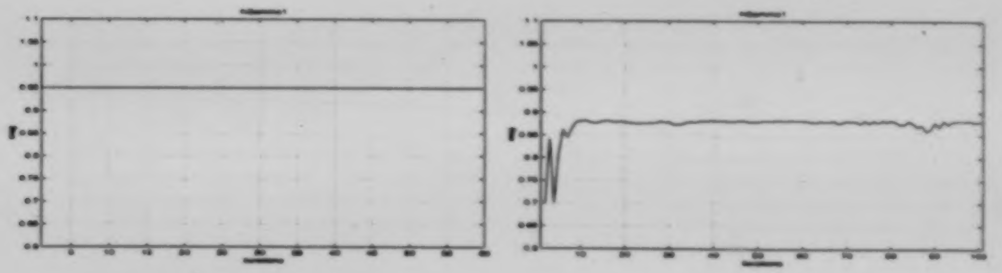
Figure 216: Constant and Variable Forgetting Factors at output1 in output layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

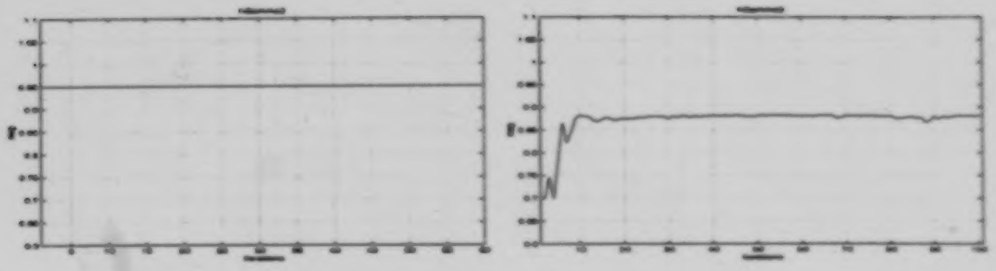
Figure 217: Constant and Variable Forgetting Factors at output2 in output layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

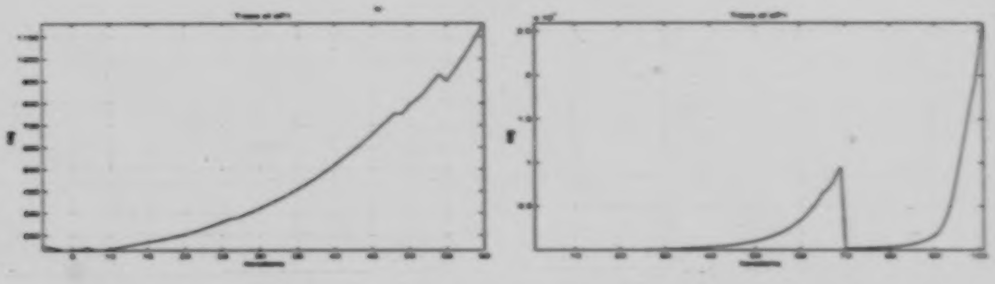
Figure 218: Constant and Variable Forgetting Factors at output1 in hidden layer



(a) Conventional CFF= 0.95

(b) Proposed VFF

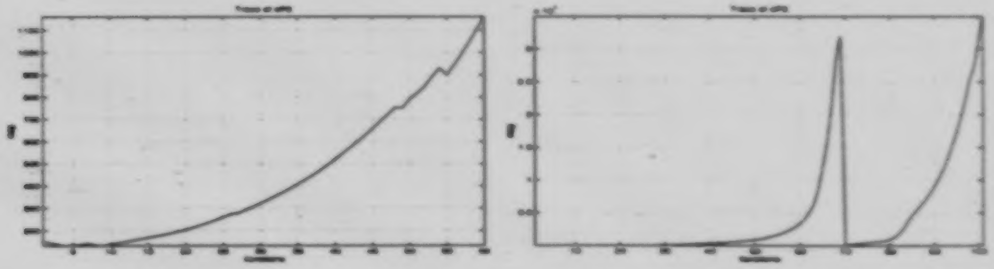
Figure 219: Constant and Variable Forgetting Factors at output2 in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

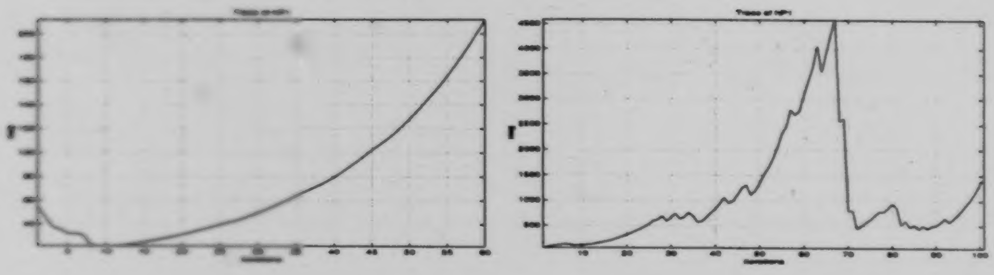
Figure 220: Trace of covariance matrices at output1 in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

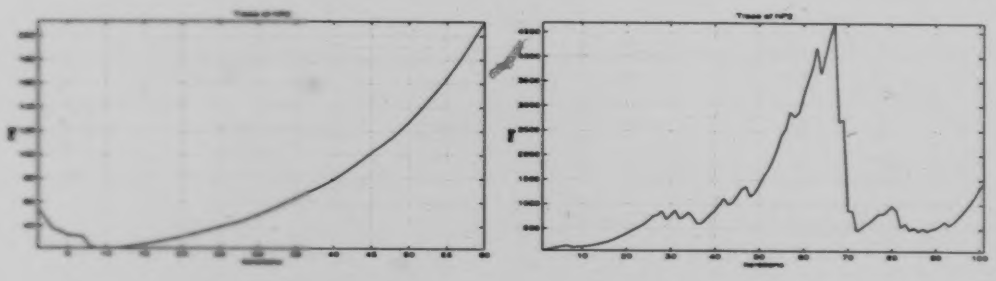
Figure 221: Trace of covariance matrices at output2 in output layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

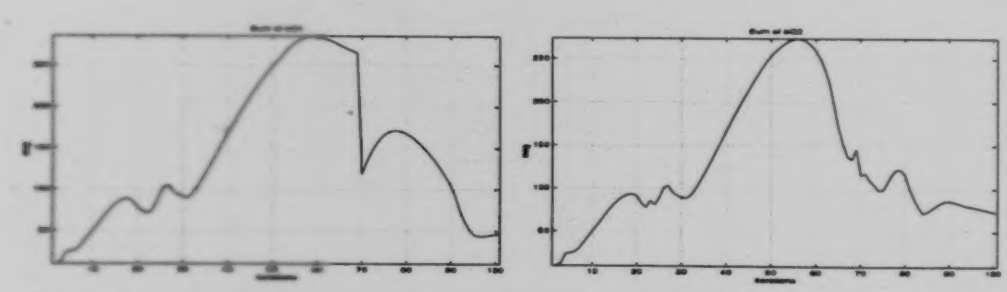
Figure 222: Trace of covariance matrices at output1 in hidden layer



(a) Periodic resetting with CFF

(b) Periodic resetting with VFF

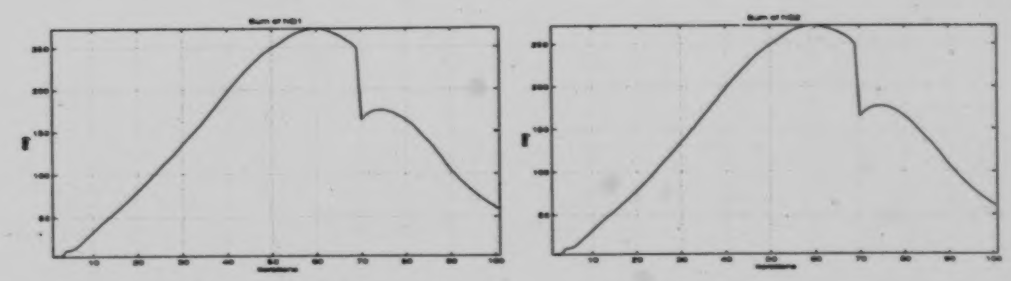
Figure 223: Trace of covariance matrices at output2 in hidden layer



(a) At Output 1

(b) At Output 2

Figure 224: Sum of Q elements (proposed) in *output layer*



(a) At Output 1

(b) At Output 2

Figure 225: Sum of Q elements (proposed) in *hidden layer*

With regard to simulation results of MIMO plant 5 with initial weights set 4 (random number), the clear clue is not found for the conventional method being wound-up, since at least, the trace of the covariance matrices in the output and hidden layers looks *alert* as shown in (a) Figures 220, 221, 222, 223. One possibility for the burst phenomenon in this case (conventional) is put on its lack of capability to recover the lost information occurring from periodic resetting of the covariance matrix with a CFF (constant forgetting factor).

6.8 Remarks on Simulation Results

In this chapter, simulation results show the comparative performance of neural system identification between the conventional and developed methods, which have been tested on five nonlinear plants with four initial synaptic weight sets including a randomly-initialized weight set respectively. The developed methods represent fast, accurate and robust neural system identification with 100% success rate while the conventional method achieved 35%. Simulation results indicated that conventional methodology (in recurrent neural network architecture and its training algorithm) turns out to be prone to incompetency such as:

1. Susceptibility to initial synaptic weights.

2. Lacking capability of keeping the covariance matrix *alert* regardless of persistent excitation condition for the input vector.

3. Less robustness about the lost information occurring in process of periodically resetting of the covariance matrix.

4. Insufficiency of using the exponential-weight constant forgetting factor to cope with various plants under consideration.

5. Inaccuracy of the identification error of nonlinear dynamic systems.

The developed neural architecture and training algorithms show clear and present performance improvement for the on-line system identification. These results will be useful for a wide range of adaptive signal processing applications. For example, FES (functional electrical stimulation) control requires good system identification techniques [1] because a neuromuscular system cannot exactly be characterized by mathematical modeling methods due to lots of dynamic uncertainties (structured and unstructured) in human body. The off-line supervised learning approach may not be adequate for the FES control application because it suffers from the same uncertainty problem after learning by a typical training set [2]. The fast, accurate and robust on-line neural system identification techniques developed in this thesis have a wide range of application areas where various dynamic uncertainties are inevitable such as underwater robotic vehicles, neuromuscular system, free-floating robot manipulators with inherently unclear mass-related properties, etc..

Bibliography

[1] R.E. Kearney and R.F. Kirsch. System identification and neuromuscular modeling. *Catholic University of America, Division of Biomedical Engineering*, 1997. Web site: http:www.ee.cua.edu.

[2] T.A. Thrasher, F. Wang, and B.J. Andrews. Self-adaptive neuro-fuzzy control of neural prostheses using reinforcement learning. *Proc. Ann. Int. Conf. IEEE/EMBS*, vol.18, 1996.

[3] D.A. Winter. *Biomechanics and Motor Control of Human Movement.* John Wiley & Sons, New York, 1990.

Chapter 7

Application of Neural System Identification To Adaptive Control Processing

7.1 Introduction

The main objective of this chapter is to show how an on-line neural system identification is synthesized with the optimal LQ (linear quadratic) control processing leading to an adaptive self-tuning control scheme. The neural network based adaptive control is expected to exhibit desirable properties such as no restrictions about system linearity, robustness with respect to different variable trajectories, good capability for uncertainties including sudden dynamic changes in the middle of an excursion, reasonable noise rejection ability, and no pre-information about a system under consideration. This is tested through the scenario of picking up a moving target (an unknown payload) by the PUMA robot manipulator without using the mathematical robotic dynamics (i.e., computed torque method). The robotic dynamics has complicated characteristics of a time-variance, high nonlinearity, strong coupling-effects among joints, and structured and unstructured uncertainties, which are all undesirable factors for the controller design. Since the overall control scheme assumes a plant being structured as a black-box in this thesis, the same control method can be applied to other unknown nonlinear systems. This is one of the most distinct features between the black-box model based control and mathematical model based control. If the mathematical dynamics of a system is exactly known in advance, the control design tasks will be much easier and the performance could be accurate. However, this classical method has limitations of an *ad-hoc* controller, short competency about uncertainties, and less feasibility for a real-life environment. Many adaptive controls [20], [18], [17] [1], [16] [7], [8], [11], [3], [4], [14] have been developed to resolve these kinds of problems. These adaptive control schemes have common components of identification and control blocks to cope with aforementioned difficulties on the control aspect. Generally speaking, there are different features between before and after 1990 research as follows:

- before 1990:
    1. Use of linear parametric models (e.g., ARX clone) for the identification.
    2. Use of adaptive self-tuning regulators (STR) instead of a tracker.
    3. Test on mostly time-invariant and deterministic systems.
    4. Lack of compensating uncertainties.

Therefore, these methods can be prone to being less adaptive for highly nonlinear time-varying systems having dynamic uncertainties for the purpose of a tracking control. As a result robot control frequently fails to work for different reference profiles of the position and velocity, especially at the high velocity [19]. To circumvent this kind of incapability, [9] and [10] suggested the combination of a variational (adaptive part) and a nominal (computed torque method) controls. The nominal control uses the direct calculation of manipulator dynamics along the desired trajectory, which requires full information of the plant dynamics [5]. The drawback on these controls is indicated as an *ad-hoc* control working for a restricted plant, reference signal and situation. The direct reason for this is postulated by the less robust identification part. For a more intelligent adaptive (control) performance, the neural networks have actively been used after 1990 based on the expectation that they can have superior capabilities for the nonlinear signal processing to the conventional methodology. However, there exist difficulties to alloy the relatively less matured ANNs techniques into the individual application as follows:

- after 1990:
  1. Unclear optimum ANNs architecture for an application.
  2. Insufficient learning capabilities in terms of the speed, the sensitivity about learning parameters, and the side-effects in each training strategy.
  3. Unclear effective-implant of ANNs results to the application.
  4. Difficulties for guaranteed stability.

In this thesis, the developed SERNN (supervision & error recurrent neural network) architecture and MRLS (modified recursive least-squares) training algorithm are combined into the control block leading to the development of the adaptive optimal LOQ (linearly-observed quadratic) self-tuning control. The nonlinear system consisting of a robot arm is used to simulate the overall performance for a tracking control without using the pre-information about plant. Since the system identification procedures and results have been shown in Chapters 4, 5, and 6, this chapter presents a nonlinear tracking control law by use of LOQ optimal techniques based on the on-line system identification result. A system with uncertainties is assumed as a *black-box* by the trained neural network model after system identification. Hence, the control is carried out through (a) the on-line neural synaptic weights training, (b) the mathematical formulation of the neural model output with learned weights, (c) the model-based optimal control law, derived by replacing a real system with the mathematical input and output equation depending on the neural architecture.

The ***model-only based adaptive control*** has the advantage when no physical pre-knowledge about a system being controlled is available or cannot be used in the process of control design. Therefore, the control design is carried out depending only on measurements while the designer does not know how the internal mechanism in the real system converts inputs to outputs. Therefore, the system identifier plays an important role for the overall control performance. Accordingly, an identifier (or estimator) having fast, accurate, robust and generalized on-line estimation capabilities is required for the on-line optimal control synthesis procedure. This kind of control is attractive under situations:

- (a) when the derivation of complex mathematical dynamics for a plant being concerned is difficult, inaccurate, or sometimes impossible,
- (b) when there exist dynamic uncertainties under unknown environments,
- (c) when more universal control is required to avoid an *ad-hoc* controller under diverse environments, and
- (d) when easy replacement of different control laws is necessary.

7.2 Adaptive Self-tuning Control

The basic architecture of an adaptive self-tuning (explicit) control (STC) is shown in Figure 226. The STC is a discrete-time method which attempts to overcome problems of model-plant mismatch by automating the overall design procedure and repeating the steps of identification and control gain during each sampling period. The STC, therefore, has the ability to tune the change of plant dynamics continuously. One notes that the system identifier replaces a real system output into the mathematical expression of the model output with estimated parameters. In the adaptive STC case, the on-line control synthesis block treats the estimated model (or trained neural network) as if it were the true plant, which is leading to the model following control. This strategy is generally known as *certainty equivalence* adaptive control.

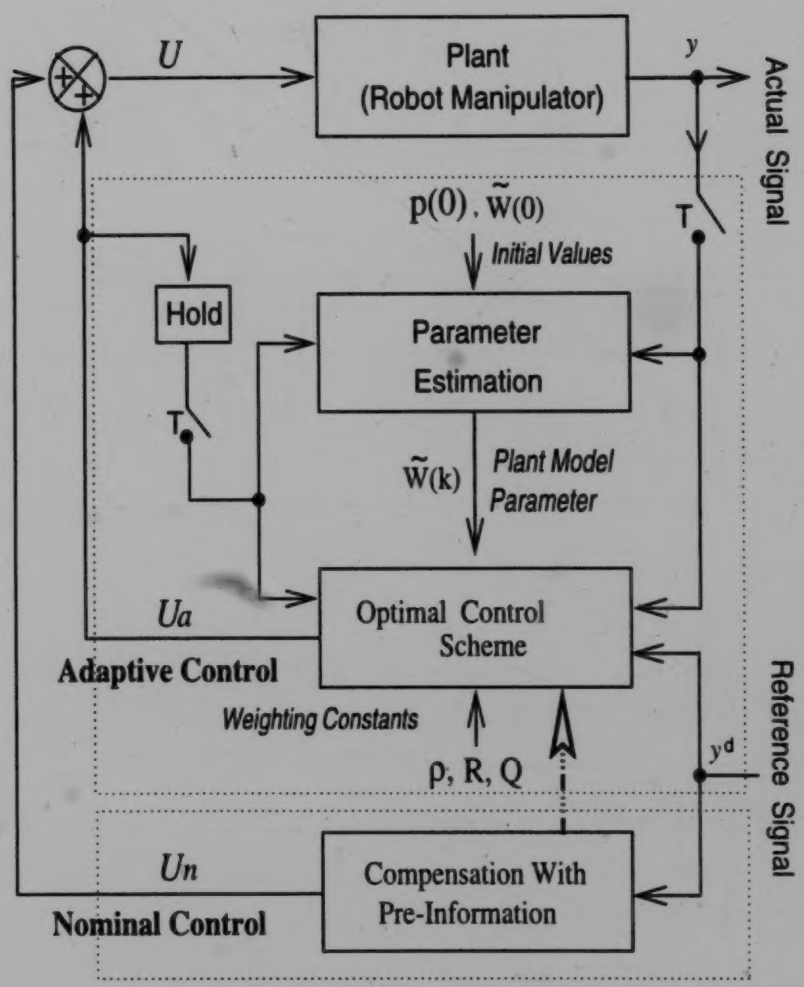

+
+
U
Plant
(Robot Manipulator)
y
Actual Signal
p(0), W̃(0)
Initial Values
T
Hold
Parameter
Estimation
T
W̃(k)
Plant Model
Parameter
Ua
Optimal Control
Scheme
Adaptive Control
Weighting Constants
ρ, R, Q
y^d
Reference Signal
Un
Compensation With
Pre-Information
Nominal Control

Figure 226: Adaptive Self-tuning Scheme

7.3 Optimal LOQ Controller Design For MIMO System

Optimal LQ (linear quadratic) control has been presented by [15], [13], [12], and [6] mostly based on the linear time-invariant state-space model. The LQ control penalizes the tracking error and control energy by means of minimizing the cost-function although it is not clear what is the most appropriate cost-function. In this section, the LQ concept is combined into the neural model which is linearly observed by inversion of the activation function. Therefore, a linearly-observed-quadratic (LOQ) cost-function is built, and an optimal LOQ control law is constructed by minimizing the position and velocity squared-errors and the energy (control signal) as specified by the weighted quadratic cost-function below:

(7.1)

$$J_k(U) = E\Big\{\Big\|\big(g^{-1}[\dot{Y}^{act}(k+1)] - g^{-1}[\dot{Y}^{d}(k+1)]\big) + \rho \cdot \big(g^{-1}[Y^{act}(k+1)] - g^{-1}[Y^{d}(k+1)]\big)\Big\|_Q^2 \\ + \big\|U(k)\big\|_R^2 \,/\, \Sigma(k)\Big\}$$

where superscripts *act* and *d* stand for the (measured) actual and desired signals respectively, $\|\cdot\|_R^2$ indicates the generalized norm with weight R, for example, $\|\cdot\|_R^2 = U^T \cdot R \cdot U$, and cost-function weighting matrices ρ, Q and R are the design parameters which are chosen in view of the operational objects. For example, ρ emphasizes the relative importance of position tracking errors over velocity tracking errors; $Y^{act}(k+1)$ and $Y^d(k+1)$ describe the measured and desired (position) trajectory vector as a sequence of discrete points respectively, where $Y^d(k+1)$ is approximately represented by $Y^d(k+1) = Y^d(k) + \dot{Y}^d(k) \cdot T_s$ because $Y^d(k+1)$ is unknown at time kT. A prediction algorithm may be used for desired signals, for example, the $Y^d(k+1)$ is predicted one sampling period early in real applications of vision feedback servoing [2], [8]; $g^{-1}[\cdot]$ is the inversion of the activation function; the expectation operation $E\{\cdot/\Sigma(k)\}$ is conditioned on the available measurements up to and including time kT_s. The control input $U(k)$ is *admissible* when it is a function of available measurements up to and including time $k \cdot T_s$.

The problem is to minimize the cost-function with respect to the admissible control while satisfying the constraint equation for the measured output on the neural model. The measured joint velocities of the robot arm are taken as the output variables of the proposed neural model. These actual velocities can be measured by tachometer or calculated from adjacent positional readings. The neural output equation based on the SERNN model (output layer) is written from the velocity measurements as follows:

(7.2)

$$g^{-1}\left[\dot{Y}^{act}(k)\right] \approx A^T(k)\cdot\dot{Y}^T(k-1)+B^T(k)\cdot U(k-1)+C^T(k)\cdot\mathcal{E}^{mo}(k-1)+\mathcal{E}^{mo}(k)$$

The unknown synaptic matrices $A(k)$, $B(k)$ and $C(k)$ are estimated by the on-line neural network training in the identification block as shown in the Chapter 6. Equation 7.2 can be rewritten as one sampling period advanced form as equation 7.3.

(7.3)

$$g^{-1}\left[\dot{Y}^{act}(k+1)\right] \approx \hat{A}^T(k)\cdot\dot{Y}^T(k)+\hat{B}^T(k)\cdot U(k)+\hat{C}^T(k)\cdot\mathcal{E}^{mo}(k)+\mathcal{E}^{mo}(k+1)$$

The solution for the discrete LOQ control law is obtained by substituting equation 7.3 into the cost-function equation 7.1 and by minimizing the resulting equation with respect to control gain $U(k)$. The resulting controller is expressed as follows:

(7.4)

$$\begin{aligned}U^*(k) = [R+\hat{B}\cdot Q\cdot\hat{B}^T]^{-1}\hat{B}\cdot Q\cdot\Big\{&\dot{Y}^d(k+1)-\hat{A}^T(k)\cdot\dot{Y}^{act}(k)-\hat{B}^T(k)\cdot\mathcal{E}^{mo}(k)\\ &+\rho\cdot\left(g^{-1}[Y^d(k+1)]-g^{-1}[Y^{act}(k+1)]\right)\Big\}\end{aligned}$$

where the weight matrices are selected such as $\rho=\rho^T=diag[\rho_1\cdots\rho_P]^T$, $R=R^T\geq 0$ and $Q=Q^T>0$; $\hat{A}(k)\in\Re^{P\times P}$, $\hat{B}(k)\in\Re^{P\times P}$, $\hat{C}(k)\in\Re^{P\times P}$ represent the the estimated synaptic weight matrices at sampling instant $k\cdot T_s$. Only measurements of joint velocity are assumed to be available because the measurement of joint position can be computed by integrating joint velocity measurements in the controller unit provided that the initial (rest) position $Y^{act}(0)$ is known:

(7.5)

$$
\begin{aligned}
Y^{act}(k+1) &\approx Y^{act}(k) + \dot{Y}^{act}(k) \cdot T_s \\
&\approx Y^{act}(k-1) + \dot{Y}^{act}(k-1) \cdot T_s + \frac{1}{2}\ddot{Y}(k-1) \cdot T_s^2 + \dot{Y}^{act}(k) \cdot T_s \\
&\approx Y^{act}(k-1) + \dot{Y}^{act}(k-1) \cdot T_s + \frac{1}{2}\Big[\frac{\dot{Y}^{act}(k) - \dot{Y}^{act}(k-1)}{T_s}\Big] \cdot T_s^2 \\
&\quad + \dot{Y}^{act}(k) \cdot T_s \\
&= Y^{act}(k-1) + \frac{1}{2}\dot{Y}^{act}(k-1) \cdot T_s + \frac{3}{2}\dot{Y}^{act}(k) \cdot T_s
\end{aligned}
$$

where $Y^{act}(k-1)$, $\dot{Y}^{act}(k-1)$, $\dot{Y}^{act}(k)$ are all available at present instant k.

In equation 7.3, $g[\cdot]$ is the hyperbolic tangent activation function with slope and saturation constants a, b. Its inverse function is given as follows:

$$g^{-1}[y] = v = \frac{1}{a} \cdot \ln\left(\frac{b+y}{b-y}\right) \quad \text{for } y \triangleq b \cdot \tanh\left(\frac{a}{2} \cdot v\right),\ a \neq 0,\ b > y$$

Figure 227 presents the adaptive self-tuning control scheme based on the fast online neural system identification, where the recurrent neural architecture (SERNN), MRLS training algorithm, and LOQ optimal law developed in this thesis are used for a moving-target tracking task.

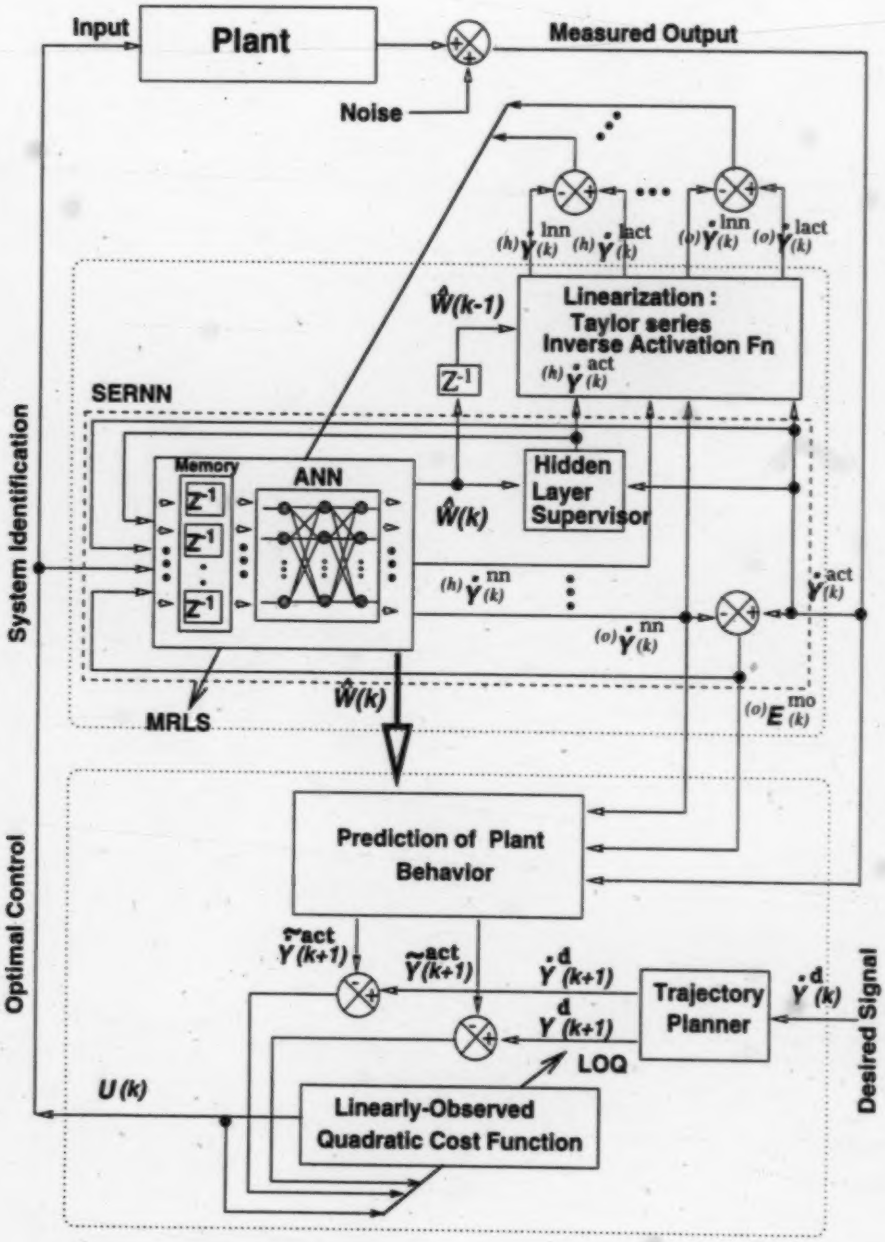

Input
Plant
Measured Output
Noise
System Identification
SERNN
Memory
ANN
Z^{-1}
Hidden Layer Supervisor
$\hat{W}(k)$
$\hat{W}(k-1)$
Linearization : Taylor series Inverse Activation Fn
$^{(h)}\dot{Y}^{act}_{(k)}$
$^{(h)}\dot{Y}^{lnn}_{(k)}$
$^{(h)}\dot{Y}^{lact}_{(k)}$
$^{(o)}\dot{Y}^{lnn}_{(k)}$
$^{(o)}\dot{Y}^{lact}_{(k)}$
$^{(h)}\dot{Y}^{nn}_{(k)}$
$^{(o)}\dot{Y}^{nn}_{(k)}$
$\dot{Y}^{act}_{(k)}$
$^{(o)}E^{mo}_{(k)}$
MRLS
Optimal Control
Prediction of Plant Behavior
$\tilde{\dot{Y}}^{act}(k+1)$
$\tilde{Y}^{act}(k+1)$
$\dot{Y}^{d}(k+1)$
$Y^{d}(k+1)$
Trajectory Planner
$\dot{Y}^{d}(k)$
Desired Signal
LOQ
U (k)
Linearly-Observed Quadratic Cost Function

Figure 227: Adaptive Self-Tuning Scheme Based on Neural Identification

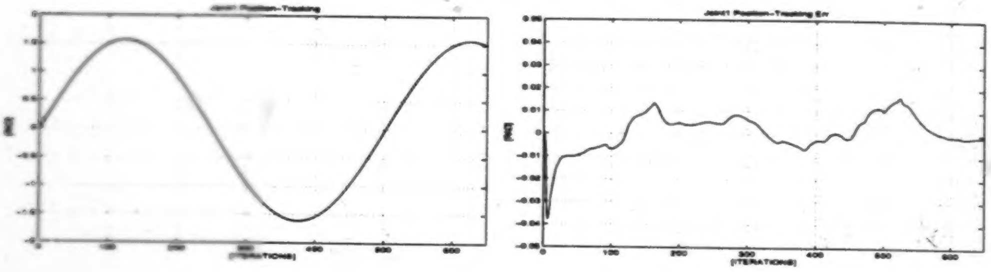
7.4 Simulation Results

The proposed neural-model based STC is tested on the simulated robot arm motion (three d-o-f) which implies forward dynamics matter. The kinematic parameters and the closed forms of $\mathbf{D}(\cdot) \in \Re^{3\times3}$, $\mathbf{C}(\cdot) \in \Re^{3\times1}$ and $\mathbf{G}(\cdot) \in \Re^{3\times1}$ in Lagrange-Euler robotic forward dynamics equation are given in the Appendix A. The optimal control input vector $\mathbf{U}(k) \in \Re^{3\times1}$ represents three generalized torques to joint 1, 2 and 3 which are computed on-line at every sampling instant k. The following two sections show two tracking scenarios to grasp a moving target:

1. with the assumption that there is no (velocity) measurement noise and no abrupt dynamic change such as picking up the payload at a random time, and
2. when measurement noise is added by a random number generator with zero-mean Gaussian distribution, the robot end-effector picks up the 2.3 Kg payload at a random time, and different desired trajectories are applied without adjusting the design parameters.

Both control simulations count on only measurement information without using the pre-knowledge on robot dynamics such as symmetry of inertia matrix, linear-parameterization property based on the closed form of the robotic dynamics equation, off-line pre-training, etc.. Both situations result in good tracking performances for variable position and velocity references.

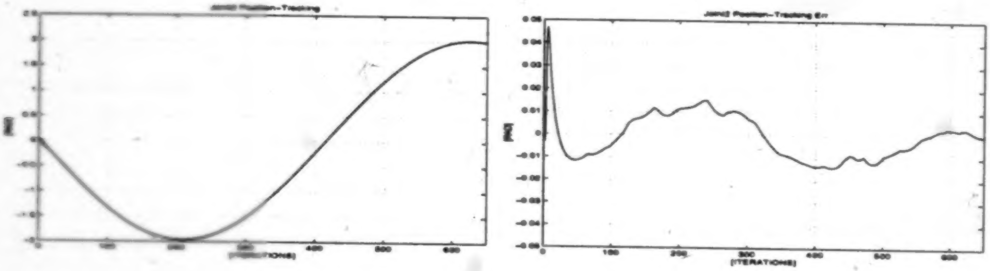
7.4.1 Without Measurement Noise and Abrupt Dynamic Change



(a) Joint1 Position Tracking (desired + actual)

(b) Joint1 Position Tracking Error

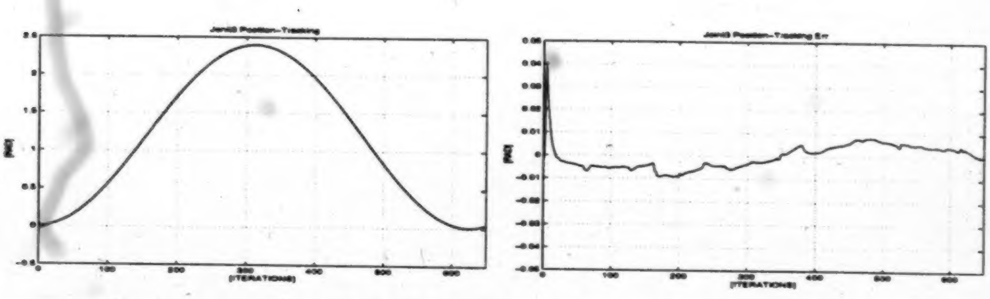
Figure 228: Joint1 Position Tracking and Error



(a) Joint2 Position Tracking (desired + actual)

(b) Joint2 Position Tracking Error

Figure 229: Joint2 Position Tracking and Error

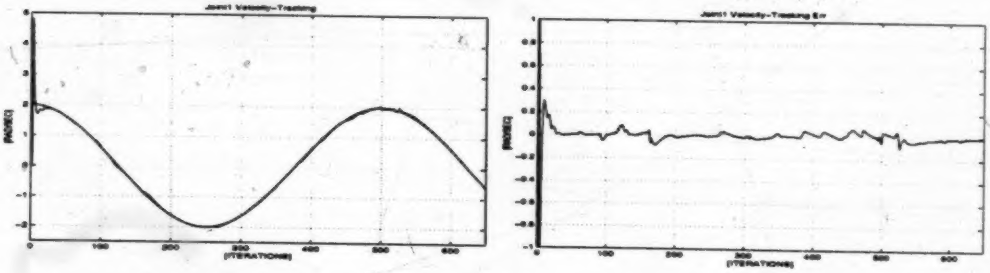


(a) Joint3 Position Tracking (desired + actual)

(b) Joint3 Position Tracking Error

Figure 230: Joint3 Angular Position Tracking and Error

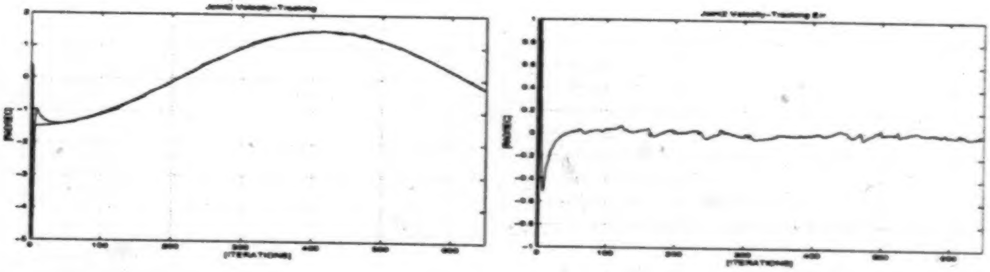
Figures 228, 229, and 230 present angular position tracking performances between desired and actual position profiles, while Figures 231, 232, and 233 show the angular velocity tracking ability. Maximum angular velocity of 2.0[rad/sec] in Figure 232 is assumed as being fast. Neural system identification results to characterize the measured velocity are shown in Figures 234, 235, and 236, which are the internal on-line calculation processes, but their performance directly influences the overall adaptive control performance. A neural position identification to characterize the position information is not required to control the position in this architecture.



(a) Joint1 Velocity Tracking (desired + measured)

(b) Joint1 Velocity Tracking Error

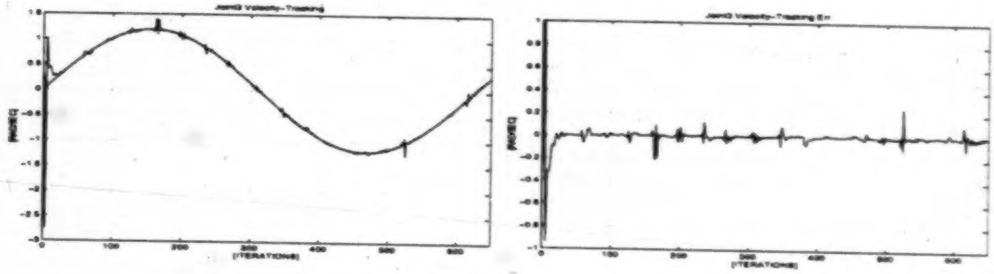
Figure 231: Joint1 Angular Velocity Tracking and Error



(a) Joint2 Velocity Tracking (desired + measured)

(b) Joint2 Velocity Tracking Error

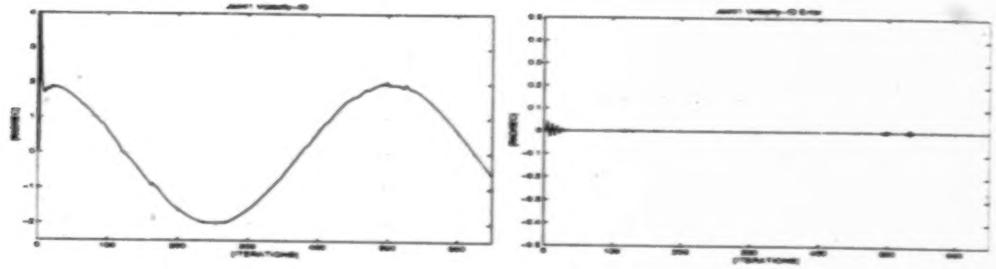
Figure 232: Joint2 Angular Velocity Tracking and Error



(a) Joint3 Velocity Tracking (desired + measured)

(b) Joint3 Velocity Tracking Error

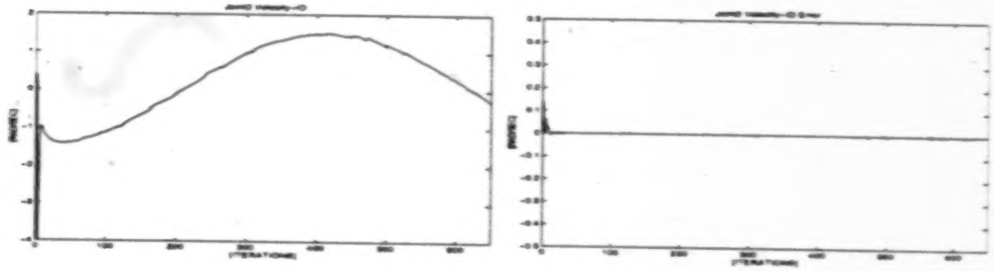
Figure 233: Joint3 Angular Velocity Tracking and Error



(a) Joint1 Velocity Identification

(b) Joint1 Velocity Identification Error

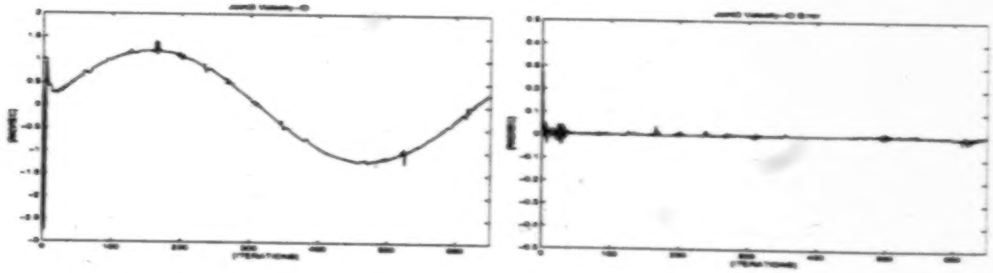
Figure 234: Joint1 Angular Velocity System Identification and Identification Error



(a) Joint2 Velocity Identification

(b) Joint2 Velocity Identification Error

Figure 235: Joint2 Angular Velocity System Identification and Identification Error



(a) Joint3 Velocity Identification

(b) Joint3 Velocity Identification Error

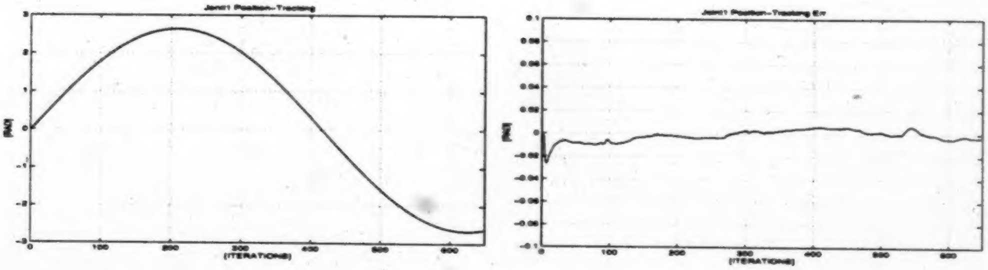
Figure 236: Joint3 Angular Velocity System Identification and Identification Error

7.4.2 Addition of Measurement Noise and Abrupt Dynamic Change On Different Trajectories

In this simulation, velocity measurement noise of $\sigma^2 = 0.01$ [rad/sec] has been added to each joint. In the middle of the excursion, the robotic end-effector has picked the payload 2.3 Kg at the random instant $k = 300$ equivalent to 3 seconds (at $T_s = 0.01$ [sec]) after starting moving. This is to simulate the sudden dynamic change. (The mass of payload has been added to that of the third link after $k = 300$.) Different desired profiles for the position and velocity are also applied. These simulation results show good compensation ability against quickly changed environments although the tracking performance is somewhat detracted from the measurement error. The initial tracking speed is fast fallen into the range of 0.3 [sec]. Since the black-box, model-based control only counts on measurement information about a system, it is intrinsically prone to be sensitive to measurement noise. This drawback has been resolved to some good extent by modifying the exponential-weight variable forgetting factor (EW-VFF) developed in the Chapter 5 as follows:

$$\begin{aligned} \gamma_j(k) &= \frac{c}{c + \left[\frac{\frac{1}{3}(\dot{y}_j^{act}(k)+\dot{y}_j^{nn}(k)+\dot{y}_j^{d}(k)/u_j(k)}{\frac{1}{3}(\dot{y}_j^{act}(k-1)+\dot{y}_j^{nn}(k-1)+\dot{y}_j^{d}(k-1))/u_j(k-1)}\right]^2} \\ &\approx \frac{c}{c + \left[\frac{(\dot{y}_j^{act}(k)+\dot{y}_j^{nn}(k)+\dot{y}_j^{d}(k))\cdot u_j(k-1)+\epsilon}{(\dot{y}_j^{act}(k-1)+\dot{y}_j^{nn}(k-1)+\dot{y}_j^{d}(k-1))\cdot u_j(k)+\epsilon}\right]^2} \end{aligned} \tag{7.6}$$

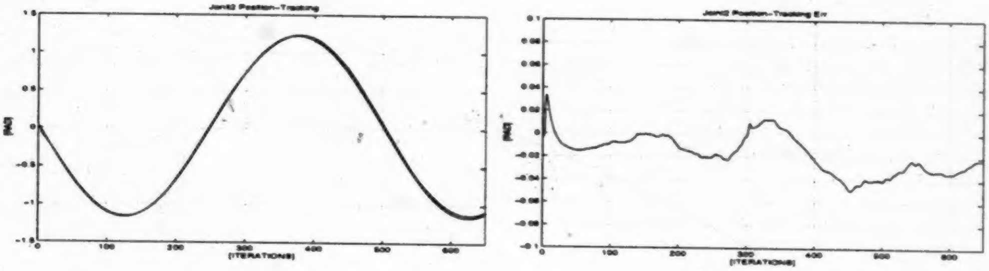
where ϵ is a small number to prevent the value from going to zero. In equation 7.6, the average of measured actual ($\dot{y}_j^{act}(k)$: corrupted by noise), internal neural ($\dot{y}_j^{nn}(k)$: no noise), and desired ($\dot{y}_j^{d}(k)$: no noise) outputs is used for the Cauchy function variable of the EW-VFF to alleviate the measurement noise sensitivity, since three values are to be same theoretically. This is in effect a low pass filter to get rid of noise which is generally known as having high frequency. The following simulation results reasonably justify aforementioned design objectives for a pure-model based nonlinear control.



(a) Joint1 Position Tracking (desired + actual)

(b) Joint1 Position Tracking Error

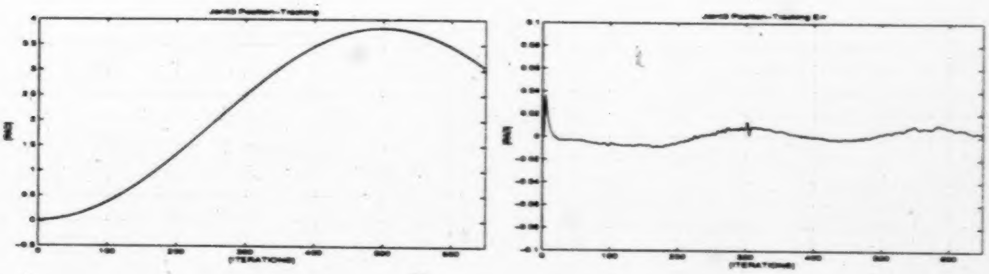
Figure 237: Joint1 Position Tracking and Error



(a) Joint2 Position Tracking (desired + actual)

(b) Joint2 Position Tracking Error

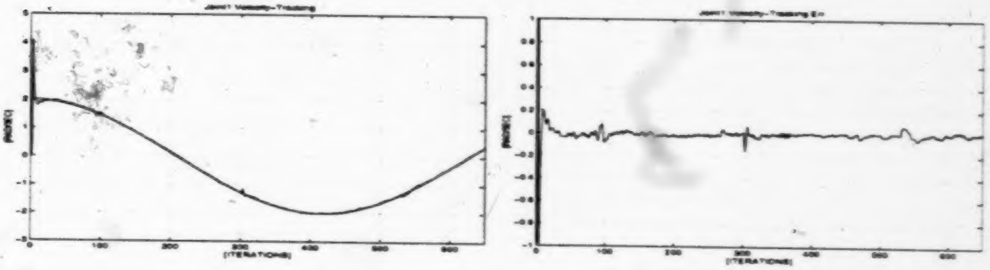
Figure 238: Joint2 Position Tracking and Error



(a) Joint3 Position Tracking (desired + actual)

(b) Joint3 Position Tracking Error

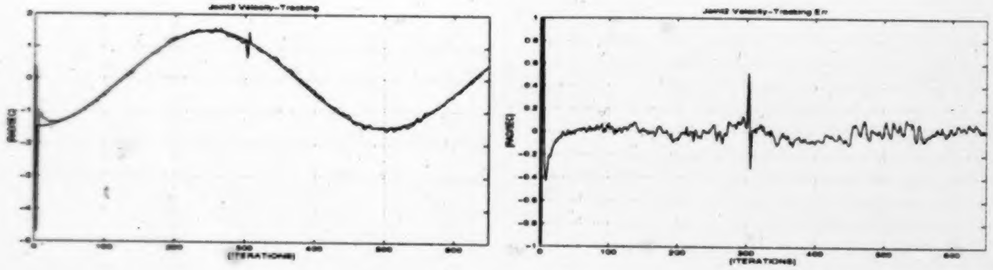
Figure 239: Joint3 Angular Position Tracking and Error



(a) Joint1 Velocity Tracking (desired + measured)

(b) Joint1 Velocity Tracking Error

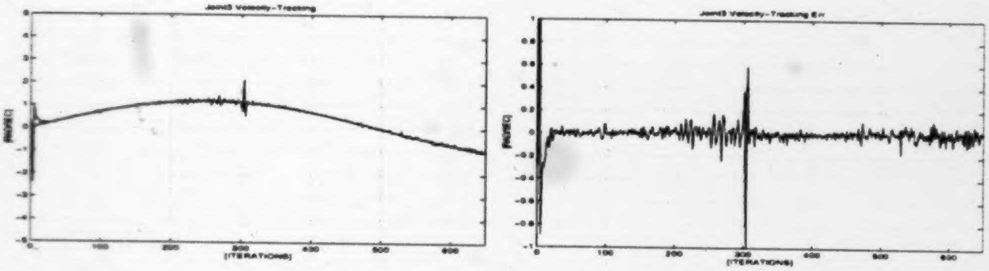
Figure 240: Joint1 Angular Velocity Tracking and Error



(a) Joint2 Velocity Tracking (desired + measured)

(b) Joint2 Velocity Tracking Error

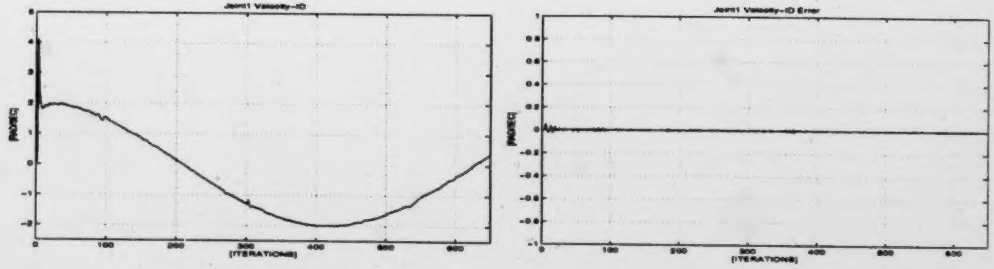
Figure 241: Joint2 Angular Velocity Tracking and Error



(a) Joint3 Velocity Tracking (desired + measured)

(b) Joint3 Velocity Tracking Error

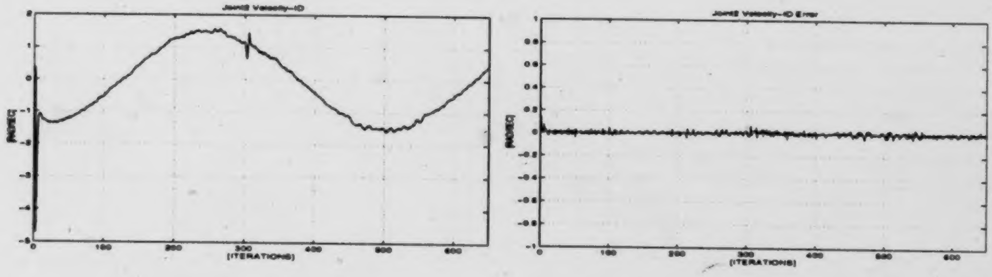
Figure 242: Joint3 Angular Velocity Tracking and Error



(a) Joint1 Velocity Identification

(b) Joint1 Velocity Identification Error

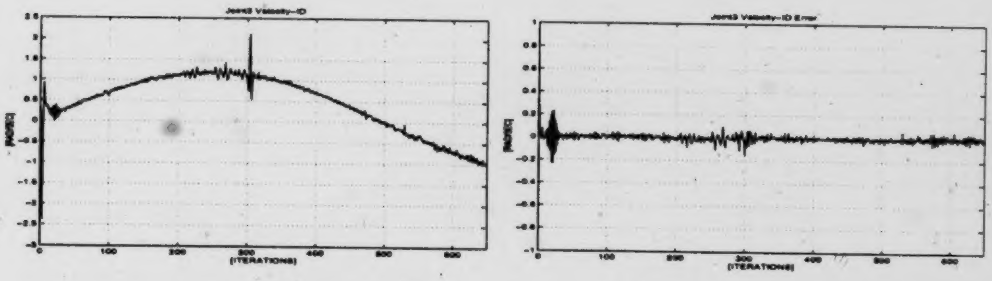
Figure 243: Joint1 Angular Velocity System Identification and Identification Error



(a) Joint2 Velocity Identification

(b) Joint2 Velocity Identification Error

Figure 244: Joint2 Angular Velocity System Identification and Identification Error



(a) Joint3 Velocity Identification

(b) Joint3 Velocity Identification Error

Figure 245: Joint3 Angular Velocity System Identification and Identification Error

7.5 Remarks

This chapter has presented the feasibility of neural system identification techniques for the adaptive control processing through robot trajectory tracking tasks under diverse references, sudden dynamics changes, with the unknown nonlinearity, time-variance, and coupling-effects. The subject of tracking control for the robot end-effector is neither new nor innovative if the pre-information can be fully characterized under a known and certain environment mostly through the mathematical dynamic equation with known kinematic parameters and the analysis of each nonlinear term based on the Lagrange-Euler closed form dynamics. However, the robot control (a control for a nonlinear, time-varying and coupled system with dynamic uncertainties) depending on only measurement information has high potential to achieve a universal nonlinear control for various systems. The pure-model based adaptive control without using pre-knowledge has seldom been found in the literature. For this, [4] recently presented a neuro-adaptive trajectory tracking control based on RBF neural network and extended Kalman filter technique. This made use of the intrinsic linearity (linear function of its weight) in the RBF network architecture, and the conventional RLS algorithm to avoid using the BP algorithm. The BP algorithm is prone to the slow convergence and other drawbacks as indicated there. But some weak points of their strategy are that they used an off-line method to compensate the structured and unstructured uncertainties[1] along a priori training data (pre-knowledge) generated from robot dynamic equation. They had to normalize and scale the input/output training data differently for the RLS algorithm with the unity forgetting factor and generic updating of the covariance matrix, which is not robust for other training sets especially when the persistent excitation condition is not met as shown in Chapter 6. This did not overcome the category of the pure-model based adaptive processing. This has led to large trajectory tracking error in showing generalization capability for the different desired trajectory, because time-invariant synaptic weights were used through the off-line learning strategy to avoid drawbacks of the conventional RLS algorithm as an on-line neural network trainer. But their use of the Kalman filter for the control part is worthy of attention. Meanwhile, the pure on-line and model based signal processing strategy in this thesis can be more suitable for an application to the creature like a human body, because its mathematical modeling is very difficult, and more uncertainties exist there.

---

[1] The structured uncertainties are due to uncertainties in parameter values while the unstructured uncertainties are because of unmodeled effects such as friction, joint flexibility, motor/gear backlash, external disturbances, etc..

Bibliography

[1] D.Y. Abramovitch and G.F. Franklin. On the stability of adaptive pole-placement controllers with a saturating acturator. *IEEE Trans. Autom. Control*, AC-35:pp.303–306, 1990.

[2] P. Allen, B. Yoshimi, and A. Timcenko. Real-time visual servoing. Technical Report CUCS-035-90, Dept. of Computer Science, Columbia University, New York, NY 10027, 1990.

[3] A.M. Annaswamy and S.P. Karason. Discrete-time adaptive control in the presence of input constraints. *Automatica*, vol.31(no.10):pp.1421–1431, 1995.

[4] L. Behera, M. Gopal, and S. Chaudhury. On adaptive trajectory tracking of a robot manipulator using inversion of its neural emulator. *IEEE Trans. On Neural Networks*, vol.7(no.6):pp.1401–1414, Nov. 1996.

[5] J.J. Craig, P.Hsu, and S. Sastry. Adaptive control of mechanical manipulators. *IEEE Conf. On Robotics and Automation, San Francisco*, April 1986.

[6] P. Dorato, C. Abdallah, and V. Cerone. *Linear-Quadratic Control: An Introduction.* Prentice Hall, Englewood Ciffs, NJ 07632, 1995.

[7] G. Feng, C. Zhang, and M. Palaniswamy. Adaptive pole placement control subject to input amplitude constraints. *Proc. 30th IEEE Conf. On Decision and Control*, pages 2493–2502, 1991. Brighton, U.K.

[8] A.J. Koivo and N. Houshangi. Real-time vision feedback for servoing robotic manipulator. *IEEE Trans. on Systems Man and Cybernetics*, vol.21(no.21), Jan/Feb 1991.

[9] C.G.S. Lee and M.J. Chung. An adptive perturbation control with feedforward compensation for robot manipulators. *Simulation*, vol.44(no.3):pp.127–136, 1985.

[10] C.S.G. Lee, R.C. Gonzalez, and K.S. Fu. *Robotics: Control, Sensing, Vision, and Intelligence.* McGraw-Hill Book Co., 1987.

[11] A.U. Levin and K.S. Narendra. Control of nonlinear dynamical systems using neural networks: Controllability and stabilization. *IEEE Trans. On Neural Networks*, vol.4(no.2):pp.192–206, March 1993.

[12] C.-F. Lin. *Advanced Control Systems Design.* Prentice Hall, Englewood Ciffs, NJ 07632, 1994.

[13] R.H. Middleton and G.C. Goodwin. *Digital Control and Estimation.* Prentice Hall, Englewood Ciffs, NJ, 1990.

[14] S.I. Mistry, S.-L. Chang, and S.S. Nair. Indirect control of a class of nonlinear dynamic systems. *IEEE Trans. On Neural Networks*, vol.7(no.4):pp.1015–1023, July 1996.

[15] E. Mosca, A. Casavola, and L. Giarre. Minimax lq stochastic tracking and servo problems. *IEEE Trans. On Automatic Control*, vol.35(no.1):pp.95–97, Jan 1990.

[16] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems neural networks. *IEEE Traans. On Neural Networks*, vol.1(no.1):pp.4–27, March 1990.

[17] M. Niedzwiecki. Steady-state and parameter tracking properties of self-tuning minimum variance regulators. *Automatica*, vol.25(no.4):pp.597–602, 1989.

[18] A. Pascoal, R.L. Kosut, G.F. Franklin, D. Meldrum, and M.L. Workman. Adaptive time-optimal control of flexible structures. *Proc. American Control Conf.*, pages 19–24, 1989. Pittsburg.

[19] Won-Kuk Son. An adaptive controller for a fast six-joint robot manipulator. *Msc Thesis, Dept. of Elec. and Comp. Eng., University of Alberta, Canada*, 1993.

[20] C. Zhang and R.J. Evans. Amplitude contrained direct self-tuing control. *Proc. IFAC Symp. on identification and system parameter estimation*, pages 325–329, 1988. Beijing.

Chapter 8

Conclusions

This research presents a *fast*, *accurate*, *robust* and *generalized* on-line neural system identification technique for dynamic nonlinear systems and its application to the adaptive control processing. The neural system identification performance has been successfully tested on four diverse SISO systems and two MIMO systems using computer simulations. The speed of the developed system is confirmed in each case by the transient tracking speed at the beginning of the identification sequence. The identification accuracy was bench-marked by the steady-state error. The robustness of the developed architecture and training techniques is proved by the determination of the algorithm stability for different initial parameters (including random number) and sudden dynamic changes of the parameters. For example, when the robot arm was picking up a variable payload at a random instant. The generality of the proposed methodologies is shown by successful tests on different systems of diverse initial weight values without changing the adjustable free parameters. The improved performance was shown by comparing simulation results between existing methods and the developed/modified strategies using dynamic system identification on-line. The generality of these algorithms is important since it avoids the use of *ad-hoc* methodology for a specific system.

Another objective of this thesis was to show how the developed identification techniques can be applied to the adaptive control processing using a neural network.

The synthesis of neural system identification skills and optimal LOQ control strategy was satisfactorily carried out on-line through simulation scenarios of a moving target grasp by the PUMA robot arm. The clear composition of identification and control blocks leads to the control scheme called the adaptive self-tuning control. The robustness with respect to measurement noise, generality on different variable trajectories and compensation capability for dynamic uncertainties were successfully presented as an adaptive controller only based on measurement signals. This black-box model control has high potential and diverse advantages for various nonlinear time-varying systems with lots of uncertainties whose mathematical modeling is difficult or impossible, because it does not use the pre-information about a plant being controlled. The significance of the measurement-only based signal processing strategy shown in this thesis can become more profound when a system under consideration is a living body. The powerful system identification ability becomes the basis for the measurement-only based signal processing scheme.

8.1 Contributions and Findings

This thesis proposes and develops a new type of neural architecture called the *Supervision & Error Recurrent Neural Network* (SERNN) and its training method called the *Modified Recursive Least-Squares* (MRLS) algorithm. The synthesis of the neural system identification and optimal control law is also shown through robotic position tracking problems. The main contributions of this research are described below and their features are summarized as follows:

1. **Application of fast RLS algorithm to the training of the multi-layer RNNs**: As a parameter estimation method, the RLS clone is well known as a fast estimation tool. However, its direct application to neural training is difficult due to the **absence of the hidden-layer teaching signal**. It is only applicable for **linear parametric models**, but has a serious limitation of **numerical wind-up phenomenon**. These problems have been solved through the research presented in this thesis.

2. **Derivation of New Teaching Signal for Hidden Layers**: The hidden-layer training has so far been carried out by the BP algorithm or by its modified clone of *hill-climbing* which has various side effects. In this thesis, a clear supervisory signal in the hidden layer is derived by the use of the L_2-norm minimization technique. This result plays a major role for the RLS algorithm to be applied to the multi-layer ANNs. Although other researchers have also tried to apply the RLS clone for the neural nets training, they fail to clearly show the solution for the **absence of the teaching signal in the hidden layer**. Recently [11] and [9] applied the Kalman filtering method to the output layer training and the BP algorithm to the hidden layer training. The shortcomings of the BP algorithm still remain. [3] applied the RLS algorithm with unity forgetting factor to a Radial Basis Function (RBF) network where the hidden layer training is unnecessary because all the hidden layer weights have unity values. The RLS algorithm here was for training only of the output layer which does not have the activation function automatically leading to the linear parameter formula for the output layer. Moreover, these works ignore issues of the forgetting factor selection and numerical burst-phenomenon.

3. **Effective Linearization Techniques**: The Taylor series linearization expansion techniques with respect to weights at the previous sampling instant and the linear observation by the inversion of the activation function are used for the ANNs training by the RLS algorithm. The Taylor series expansion performed on the recursive parameter form in this thesis leads to a fairly accurate linearization method by using all terms occurring in the first-order differentiation. The latter method is preferred because of its computational simplicity although no performance difference between the two methods was noticed.

4. **Derivation of Modified RLS Algorithm**: By alleviating the restriction to Gaussian zero-mean noise on the modeling error in the Standard RLS (SRLS) algorithm, the Modified RLS algorithm called the MRLS algorithm developed in this research improves the speed, accuracy and robustness of the neural network

for on-line system identification. The derivation processes and performance improvement are shown through simulation results in Chapters 5 and 6.

5. **Development of Exponential-Weight Variable Forgetting Factor**: This thesis developed an exponential-weight *variable* forgetting factor (EW-VFF) to cope with diverse and dynamic environments for the MRLS algorithm. The developed algorithm makes use of the shape of the Cauchy function profile based on outputs measurements. This research uses the Cauchy function's characteristics in an adaptive manner which has not been reported in the literature. The performance improvement is compared with the exponential-weight *constant* forgetting factor (EW-CFF) in terms of the identification error as shown in the Chapter 5. The combination of the EW-VFF and periodic resetting of the covariance matrix turned out to be very robust with respect to even non-persistent excitation condition.

6. **Periodic Resetting of Covariance Matrix**: When persistent excitation is not met for the regression (input) vector, the combined effect of periodic resetting of the covariance matrix and EW-VFF developed in this thesis prevented the MRLS from going to *wind-up* phenomenon as shown in the Chapter 5. This is a major drawback of all the RLS clones. The periodic resetting method turns out to be more robust with MRLS algorithm than with conventional SRLS algorithms. This robustness has been also shown in Chapter 6.

7. **Decomposition of One Layer into Three Parts**: During this research one layer of the ANN was divided into three components called the *input nodes* (IN), the *(internal) forward synaptic weights* (FW-SW) and the *output nodes* (ON), i.e., neurons. This decomposition method makes the neural net architecture easily visible, and contributes to the creation of systematic and diverse neural links. In the conventional architecture, the neuron outputs in the previous hidden layer are simply connected to the neurons in the next output layer through (internal) FW-SW. This makes it look as if there are no other data points between the prior and posterior layers, which may always result in same

number of hidden-layer neurons and IN in the next layer by easily discarding additional input nodes (points) from remote layers. By introducing the definition of IN, more data points can effectively be found inside multi-layer ANNs. For example, both the Jordan and the Elman's RNNs do not have the recurrent connections in the middle layers, where new and meaningful connections might be able to be developed by introducing the IN. Besides, the decomposition method enables multi-layer SERNN to have a systematic calculation of the number of synaptic weights and IN for output and hidden layers as follows.

For Output Layer:

$$\begin{aligned}(\text{Number of IN}) &= 2 \times P + N \\ &= 3 \times P \quad \text{for } N = P : \text{a best supervisor in hidden-layer}\end{aligned}$$

$$(\text{Number of Weights}) = (2 \times P + N) \times P$$

For Hidden Layer:

$$(\text{Number of IN}) = 2 \times N + M$$

$$(\text{Number of Weights}) = (2 \times N + M) \times N$$

where P, N and M are the number of outputs in the output layer, hidden neurons and external inputs in the input layer respectively for the system under consideration. Since P and M are determined to be same as the number of inputs and outputs for a real plant under consideration, the N becomes the only architectural free (variable) parameter. N was determined to be the same as P without loss of general performance through this research. If the size of the proposed neural architecture (SERNN) is not enough for the required performance as an identifier, then N can be selected such that $N > P$. Meanwhile, there are no theoretical limits on the number of hidden layers, but usually in practice there will be only one or two hidden layers. This research uses two-layer SERNN without loss of performance. This resolves the issue about ANNs' size determination to some extent when the size of networks is defined by the number of input nodes in the hidden and output layers, the number of output neurons (N, P) per layer and the number of layers.

8. **Use of Feedforward Inter-layer Synaptic Weights**: Another contribution from this research is the introduction of feedforward inter-layer synaptic weights to the RNNs. This creates meaningful additional-connection from the hidden layers to the output layer by forcing the input signals in the hidden layer not to go through the (internal) forward synaptic weights in the middle layer. This enables an input signal to go through at least two channels:

   - (a) internal synaptic weights in each middle layer.
   - (b) inter-layer synaptic weights which is making an input to skip a certain middle layer(s).

   Therefore, this makes use of the direct influence of the (external) inputs in the hidden-layer to the output layer especially when each layer has (unity) time-delay units for the memory. This leads to wider time-order in the context of time difference equation. This may also simplify the inverse process of a neural network to derive the input signals given the (desired) neural network output signals with known inter-layer synaptic weights after training. One application of these connections may be found in deriving the control law on-line effectively (i.e., external input signal to neural identification nets) called the *inverse neural control*, where the desired variable-output signals are given with the trained net as a neural emulator for a plant being controlled.

9. **New Neuron Model for Automatic Bias Selection**: A further contribution involves the modification of a conventional neuron model. Although the *bias* value in an artificial neuron model plays an important role as shown in the Chapter 4, not enough research has been carried out to show its determination. [2] showed that the *bias* keeps a constant level of activation in the absence of input and must be so chosen to avoid a dead zone. [5] stated that the introduction of the bias to the neural network design will contribute to mean-square error between the desired and neural responses. [8] suggests the use of a constrained network architecture where the constraints and, therefore, the bias

may take the form of prior knowledge built into the network design. However, a global methodology to determine its value and its effect on the bias does not seem to exist to date. In a general neuron model, the *threshold* (−1.0) or *bias* (+1.0) is applied for lowering and increasing the net input of the activation function respectively. This may be seen as the manual *trial and error* method for the *ad-hoc* application purpose, which causes difficulty in determining its magnitude and sign and to analyze their eligibility for generalization. The new neuron model in this thesis has a topology such that:

- (a) the bias or the threshold takes its value from the feedback of error between supervisory signal and neuron output.
- (b) the synaptic weight is added for the above error feedback signal.

The new neuron model can make an automatic decision on the sign and the magnitude for the threshold and bias values. The error feedback can contribute to the bias/threshold implementation and to the performance improvement of neural network training in terms of speed and robustness with respect to the initial weights and the dynamic uncertainties. The idea behind the development of the new neuron model originates from the spinal motor neuron of the human brain, which have been used for most of the artificial neurons. Human neurons may have an intrinsic structure which uses a sort of error correction methodology to learn the unfamiliar environment. This error correction function results in realization of the classical feedback P-control in the new neuron model, where feedback gain P is adjusted by the added synaptic weight to the **variable bias** of recurrent error to minimize the modeling error. In the conventional neuron model, however the error correction function is not found in the model itself.

10. **Supervision Feedback**: The research presented in this thesis uses the recurrent supervision signal (external feedback) from outside the neural net for the RNNs, that is the output from the plant. In the literature, for all recurrent neural architecture, the self-feedback (or internal feedback) of the output from

hidden and/or output layers has been done. The novel recurrent architecture developed in this thesis is based on the assumption that the external feedback is able to transfer more accurate information about the system into the networks than the internal feedback can do alone. At the beginning of on-line training with poor initial weights, a neural system identifier generally is poor in the transient tracking. This poor neural output may lead to a vicious-cycle of mal-information by its self-feedback. Based on this heuristics, the feedback of the desired (supervision) signal is designed for an improved recurrent neural network architecture which makes a more robust neural system identifier with less optimal initial weights.

11. **Use of Hyperbolic Tangent for Activation Function with Slope and Saturation**: Many kinds of nonlinear activation functions for an artificial neuron exist such as the sigmoid (logistic) function, Gaussian function, Signum function (hard limiter), saturation limiter, Schmitt trigger (hard limiter with hysteresis), dead zone limiter, deadspace comparator (hard limiter with deadzone), quadratic function, absolute value function, etc.. Although the biological basis for a general activation function has not been established, most neural system identification processes are performed based on the following preferable features:

    - The Activation function (AF) is desirable to have boundedness from negative to positive for real-valued dynamic systems, unless input/output data are normalized and scaled differently.
    - The AF must be easily differentiable and invertible for a flexible training purpose.
    - The AF must have flexible parameters with features such as slope (sharpness) and saturation (magnitude). It is most desirable that the selected parameter values can be generalized for other applications or they have an adaptive variable scheme.

The research presented in this thesis uses the *hyperbolic tangent* for the activation function shown below which reasonably satisfies the aforementioned features.

$$\begin{aligned} g(v) &= b \cdot \tanh(\frac{a}{2} \cdot v) \\ &= b \cdot \left(\frac{1 - e^{-a \cdot v}}{1 + e^{-a \cdot v}} \right) \end{aligned}$$

where constants b and a control the saturation value and the slope of the activation function respectively. The fixed values were heuristically selected and worked for the on-line system identification tested on all six different nonlinear plants shown in the Chapters 5 and 6.

12. **Combination of Neural System Identification and Optimal control**: For an adaptive tracking control problem in this thesis, the optimal LOQ control law has been derived based on the developed neural system identifier leading to the adaptive self-tuning control for a nonlinear time-varying system. This control scheme satisfies the following design objectives:

    (a) It depends on only measurements of input and output signals.

    (b) It is robust with respect to different variable trajectories.

    (c) It has the compensation ability for dynamic uncertainties.

    (d) It shows less sensitivity about measurement noise.

    (e) It is easy to replace the existing controller with a different one without changing the control structure.

    The developed adaptive processing scheme is adequate for a system whose mathematical modeling is difficult or impossible.

8.2 Recommendations for Further Research Directions

This research describes the development of a fast, accurate, robust and on-line neural network for system identification, and application to adaptive tracking problem . The following work has been identified as future research.

The first area to address is to replace the constant design parameters with adaptive variables for a more flexible neural identifier. This research uses three constants for free design parameters: c, a and b are called the *Cauchy function constant* in the design of EW-VFF, the *slope* of hyperbolic tangent activation function and the *saturation value* of same activation function, respectively. Heuristically selected, these fixed values have been shown to work suitably for all simulated plants and test scenarios because their outputs still remain as variables. However, the comprehensive design for a more flexible and generalized system identifier is desired. Some suggestions are given below:

- Since c is related to the determination of exponential-weight variable forgetting factor (EW-VFF) by controlling the sharpness of the Cauchy function profile, an adaptive form of $c = f$(situation for EW-VFF through every measurements) is desired to cope with unknown systems and environments more flexibly.
- In the activation function of $b \cdot \tanh(\frac{a}{2} \cdot v)$, an *adaptive slope* $a(k)$ can be used, and expected to increase the learning speed and to improve the generalization. Similar ideas for the varying activation function are shown in [7] in conjunction with the BP learning algorithm. The *saturation value* b does not have a specific value for its optimal selection, but its value needs to avoid the normalization process of the system output under consideration and to take account of the range of real-value outputs.

In this thesis, the periodic resetting of the covariance matrix has been successfully used with the newly derived MRLS algorithm to cope with the situation of *non-persistent excitation.* The primary issue on the use of resetting method is that the training algorithm should be robust enough to recover the lost information which occurs during the periodic resetting process. The MRLS algorithm turns out to be robust with respect to the lost knowledge. The next issue is to determine the time of the periodic resetting. The determination of the resetting time has to make two decisions to keep reasonable PE conditions such as:

- (a) optimal *sampling-instance* for the *seed* covariance matrix.
- (b) optimal *interval* in resetting periodically.

In this research, the value for first resetting instance and optimal interval has been heuristically selected as $35(= k)$. This means updating the covariance matrix every 35 iterations by the covariance matrix sampled at instant $k = 35$. The value of 35 was selected based on fact that the RLS clones generally take $20 \sim 35$ iterations for steady estimates. An alternative method to determine the optimal, or variable, updating-time for the *guaranteed-alertness* of the covariance matrix could be taken up as future work. One plausible way for this is to transform the covariance matrix information (e.g. *Trace*) in time domain into frequency domain and then, to find the critical (minimum) frequency and magnitude with which the covariance matrix can always stay *alert*.

The development of an advanced neuron model is demanded for more dexterous artificial neural processing. Since the conventional neuron has been modeled after the biological one in simple manner, the introduction of the *feedback control* concept (for example, PID) to a neuron model inside may lead to more power for the advanced neural function depending on applications. The capability of the noise rejection is the one of well-known feedback control functions.

For the MRLS algorithm, a rigorous mathematical proof of its stability may be desired for its application to various disciplines. This mathematical task meets many inherent difficulties when the concerned system has features of time-variance, nonlinearity, uncertainties, etc.. The proof for stability has not yet matured enough for the neural networks research especially for a universal-purpose neural identifier. The mathematically rigorous proof of the stability for general neural nets may remain as an open-problem to neural networks researchers for the time being except for ad-hoc networks.

In the adaptive self-tuning control scheme, since the *certainty-equivalence principle* is not applicable during the initial learning phase, the input (control) signal can be infinitely large and not realizable at the beginning of training. This leads to the difficult implementation of the STC scheme with the random initialization. This is because the control block regards the neural identifier (emulator) as a real system even during the learning period. One partial remedy for this problem lets the control block take part in both training of the identification block and minimizing the control tracking error during the learning phase, and then decaying the training role after the learning phase. The key technique for this will be how the identification error model is effectively included in the control block.

Hardware development for the artificial neural networks, *microelectronic* or *optoelectronic* neural network technology has matured over the past few years. A reliable performance can be obtained from VLSI circuits under carefully controlled conditions as shown in [6], [10], [4]. The use of analogue VLSI allows low power, low cost and area efficient hardware realizations which can perform the computationally intensive neural operation at high speed. Meanwhile, *optoelectronic neural networks* may improve the performance of microelectronic neural networks limited by the capacity of their interconnections as operating speeds are increased. The use of optical connections to link electronic elements can offer a way to overcome this problem. These implementation trials are shown in [1] and [12]. This author suggests real-life applications using the neural architecture and training method developed in the thesis for future neural research. The neural system identification has found many important applications [13] in the areas of model-based adaptive control, signal processing, as well as in a variety of other fields such as medicine and bio-engineering, physical sciences, economics, and social sciences.

Bibliography

[1] T.J. Allen, K.-S. Hung, K.M. Curtis, and J.W. Orton. Practical optoelectronic neural network realizations based on the fault tolerance of the backpropagation algorithm. *IEEE Trans. on Neural Networks*, vol.7(no.2):pp.488–500, March 1996.

[2] James A. Anderson. *An Introduction to Neural Networks*. The MIT Press, Cambridge, Massachusetts, London, England, 1995.

[3] L. Behera, M. Gopal, and S. Chaudhury. On adaptive trajectory tracking of a robot manipulator using inversion of its neural emulator. *IEEE Trans. On Neural Networks*, vol.7(no.6):pp.1401–1414, Nov. 1996.

[4] G.A. Cairns and L. Tarassenko. Implementation issues on-chip learning with analogue vlsi mlps. *IEE 4th Int. Conf. on Artificial Neural Networks*, pages 465–470, 26-28, June 1995. Conference Publication No. 409.

[5] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and bias/variance dilemmma. *Neural Computation*, vol.4:pp.1–58, 1992.

[6] M. Holler, S. Tam, H. Castro, and R. Benson. An electrically trainable artificial neural network (etann) with 10240 floating gate synapses. *Proceedings of the International Joint Conference on Neural Networks*, pages 191–196, 1989.

[7] J.K. Kruschke and J.R. Movellan. Benefits of gain: speeded learning and minimal layers in back-propagation networks. *IEEE Trans. Systems, Man and Cybernetics*, vol.21:pp.273–280, 1991.

[8] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. In ed. D.S. Touretsky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404, San Mateo, CA., 1990. Morgan Kaufmann.

[9] K.-N. Lou and R.A. Perez. A new system identification technique using kalman filtering and multilayer neural networks. *Artificial Intelligence in Engineering*, vol.10:pp.1–8, 1996.

[10] A. Murray and L. Tarassenko. *Analogue Neural VLSI*. Chapman and Hall, 1994.

[11] R.S. Scalero and N. Tepedelenlioglu. A fast new algorithm for training feedforward neural networks. *IEEE Trans. on Signal processing*, vol.40(no.1):pp.202–210, 1992.

[12] R.P. Webb. Effect on noise in an optoelectronic neural networks. *IEE 4th Int. Conf. on Artificial Neural Networks*, pages 499–504, 26-28, June 1995. Conference Publication No. 409.

[13] B. Widrow, D. Rumelhart, and M. Lehr. Neural networks: Applications in industry, business and science. *Communications of the ACM*, vol.37(no.3):pp.93–105, 1994.

Appendix A

Customized L-E Dynamics Equation for Three-DOF PUMA Robot

The following equations are source codes for the MATLAB based on the Lagrange-Euler closed form of the three-joint (revolute) robotic dynamics.

(1) u: generalized input torque vector (3x1).
(2) p: angular joint-position vector (3x1).
(3) v: angular joint-velocity (3x1).
(4) acc: angular joint acceleration vector (3x1).
(5) kt: sampling instant.
(6) payload: mass of the payload (Kg).
(7) time_payload: sampling instant when picking up the payload.

```
%%%%%%%%%%%%%%%%% MATLAB CODES %%%%%%%%%%%%%%%%%%%%%%%%
function [acc]=dof3pu_nm(u,p,v,kt,payload,time_payload)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ( (size(u) ~= size(p)) | (size(p) ~= size(v)) | ...
     (size(v) ~= size(u)) )
   disp('Dimension Error1 in dof3puma.m');
   break;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Kinematic Parameters %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m6=0.17047;                               % Kg
m1=m6*33.5; m2=m6*77.3; m3=m6*36.3;       % All are [Kg].
```

```
a2=0.382; a3=0.039 ;d2=0.1375; d3=0.1235;  % All are [meter].
r2x=0.05; r3x=0.03;                        % All are [meter].
M2x=m2*r2x; M3x=m3*r3x;
k1xx=0.0451; k1yy=0.0451;k1zz=0.00579;     % unit:[meter^2]
k2xx=0.05657; k2yy=0.18470; k2zz=0.1408;   % k1xx reprsents k1xx^2
k3xx=0.06728; k3yy=0.06791; k3zz=0.0036;
J1xx=0.5*m1*(-k1xx+k1yy+k1zz);
J1yy=0.5*m1*(k1xx-k1yy+k1zz);
J1zz=0.5*m1*(k1xx+k1yy-k1zz);
J2xx=0.5*m2*(-k2xx+k2yy+k2zz);
J2yy=0.5*m2*(k2xx-k2yy+k2zz);
J2zz=0.5*m2*(k2xx+k2yy-k2zz);
J3xx=0.5*m3*(-k3xx+k3yy+k3zz);
J3yy=0.5*m3*(k3xx-k3yy+k3zz);
J3zz=0.5*m3*(k3xx+k3yy-k3zz);
GR=9.80;  % [m/sec^2]

%% Picks up a payload at sampling instant "time_payload"
if(kt == time_payload)
   disp('Robot picking up payload at :'), kt, payload
   disp('Pause: Press any key to continue...');
   pause
end
if(kt >= time_payload)
   m3 = m3 + payload;
end


p1=p(1,1); p2=p(2,1); p3=p(3,1); p23=p2+p3;
v1=v(1,1); v2=v(2,1); v3=v(3,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Inertial Coefficients (3x3): Symmetric %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% D[1,1] %%%
d3x3(1,1)= ...
  J3xx*cos(p23)^2+J3yy*sin(p23)^2+J3zz+(d3)^2*m3+2*M3x*a2*cos(p2)...
  *cos(p23)+(a2)^2*m3*cos(p2)^2+2*d2*d3*m3+(d2)^2*m3+2*M3x*a3*...
  cos(p23)^2+2*a2*a3*m3*cos(p2)*cos(p23)+(a3)^2*m3*cos(p23)^2+...
  J2xx*cos(p2)^2+J2yy*sin(p2)^2+J2zz+2*M2x*a2*cos(p2)^2+(a2)^2*...
  m2*cos(p2)^2+(d2)^2*m2+J1xx+J1zz;

%%%% D[1,2] %%%
d3x3(1,2)= ...
  M3x*d3*sin(p23)+a2*d3*m3*sin(p2)+a3*d3*m3*sin(p23)+M3x*d2*sin(p23) ...
  +a2*d2*m3*sin(p2)+a3*d2*m3*sin(p23)+M2x*d2*sin(p2)+a2*d2*m2*sin(p2);
```

```
%%% D[2,1] %%%
d3x3(2,1) = d3x3(1,2);

%%% D[2,2] %%%
d3x3(2,2) = ...
  J3xx+J3yy+2*M3x*a2*cos(p3)+(a2)^2*m3+2*M3x*a3+2*a2*a3*m3*cos(p3)+ ...
  (a3)^2*m3+J2xx+J2yy+2*M2x*a2+(a2)^2*m2;

%%% D[1,3] %%%
d3x3(1,3) = ...
  M3x*d3*sin(p23)+a3*d3*m3*sin(p23)+M3x*d2*sin(p23)+a3*d2*m3*sin(p23);

%%% D[3,1] %%%
d3x3(3,1) = d3x3(1,3);

%%% D[2,3] %%%
d3x3(2,3) = ...
 J3xx+J3yy+M3x*a2*cos(p3)+a2*a3*m3*cos(p3)+2*M3x*a3+(a3)^2*m3;

%%% D[3,2] %%%
d3x3(3,2) = d3x3(2,3);

%%% D[3,3] %%%
d3x3(3,3) = J3xx+J3yy+2*M3x*a3+(a3)^2*m3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Centrifugal and Coriolis Forces (3x1) %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% C[1,1] %%%
c3x1(1,1) = ...
 M3x*d3*cos(p23)*(v3)^2+M3x*d2*cos(p23)*(v3)^2+a3*d3*m3*cos(p23)* ...
 (v3)^2+a3*d2*m3*cos(p23)*(v3)^2+2*M3x*d3*cos(p23)*v2*v3+2*M3x*d2* ...
 cos(p23)*v2*v3+2*a3*d3*m3*cos(p23)*v2*v3+2*a3*d2*m3*cos(p23)*v2*v3 ...
 -2*J3xx*cos(p23)*sin(p23)*v1*v3-2*M3x*a2*cos(p2)*sin(p23)*v1*v3+2*J3yy* ...
 cos(p23)*sin(p23)*v1*v3-4*M3x*a3*cos(p23)*sin(p23)*v1*v3-2*a2*a3*m3* ...
 cos(p2)*sin(p23)*v1*v3-2*(a3)^2*m3*cos(p23)*sin(p23)*v1*v3+M3x*d3* ...
 cos(p23)*(v2)^2+M3x*d2*cos(p23)*(v2)^2+a2*d3*m3*cos(p2)*(v2)^2 ...
 +a2*d2*m3*cos(p2)*(v2)^2+a3*d3*m3*cos(p23)*(v2)^2+a3*d2*m3*cos(p23)* ...
 (v2)^2+M2x*d2*cos(p2)*(v2)^2+a2*d2*m2*cos(p2)*(v2)^2-2*a2*a3*m3* ...
 sin(2*p2+p3)*v1*v2-2*M3x*a2*sin(2*p2+p3)*v1*v2-2*J3xx*cos(p23)*sin(p23)* ...
 v1*v2+2*J3yy*cos(p23)*sin(p23)*v1*v2-2*(a2)^2*m3*cos(p2)*sin(p2)*v1*v2 ...
 -4*M3x*a3*cos(p23)*sin(p23)*v1*v2-2*(a3)^2*m3*cos(p23)*sin(p23)*v1*v2 ...
 -2*J2xx*cos(p2)*sin(p2)*v1*v2+2*J2yy*cos(p2)*sin(p2)*v1*v2-4*M2x*a2*cos(p2)* ...
 sin(p2)*v1*v2-2*(a2)^2*m2*cos(p2)*sin(p2)*v1*v2;
```

```
%%% C[2,1] %%%
c3x1(2,1) = ...
 -M3x*a2*sin(p3)*(v3)^2-a2*a3*m3*sin(p3)*(v3)^2-2*M3x*a2*sin(p3)*v2*v3- ...
 2*a2*a3*m3*sin(p3)*v2*v3+a2*a3*m3*sin(2*p2+p3)*(v1)^2+M3x*a2*sin(2*p2+p3) ...
 *(v1)^2+J3xx*cos(p23)*sin(p23)*(v1)^2-J3yy*cos(p23)*sin(p23)*(v1)^2+ ...
 (a2)^2*m3*cos(p2)*sin(p2)*(v1)^2+2*M3x*a3*cos(p23)*sin(p23)*(v1)^2+ ...
 (a3)^2*m3*cos(p23)*sin(p23)*(v1)^2+J2xx*cos(p2)*sin(p2)*(v1)^2-J2yy* ...
 cos(p2)*sin(p2)*(v1)^2+2*M2x*a2*cos(p2)*sin(p2)*(v1)^2+(a2)^2*m2* ...
 cos(p2)*sin(p2)*(v1)^2;

%%% C[3,1] %%%
c3x1(3,1) = ...
 M3x*a2*sin(p3)*(v2)^2+a2*a3*m3*sin(p3)*(v2)^2+J3xx*cos(p23)*sin(p23)* ...
 (v1)^2+M3x*a2*cos(p2)*sin(p23)*(v1)^2-J3yy*cos(p23)*sin(p23)*(v1)^2 ...
 +2*M3x*a3*cos(p23)*sin(p23)*(v1)^2+a2*a3*m3*cos(p2)*sin(p23)*(v1)^2 ...
 +(a3)^2*m3*cos(p23)*sin(p23)*(v1)^2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Gravitational Forces (3x1) %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% G[1,1] %%%
g3x1(1,1) = 0.0;

%%% G[2,1] %%%
g3x1(2,1) = ...
 -a3*GR*m3*cos(p23)-a2*GR*m3*cos(p2)-M3x*GR*cos(p23)-a2*GR*m2*cos(p2) ...
 -M2x*GR*cos(p2);

%%% G[3,1] %%%
g3x1(3,1) = -a3*GR*m3*cos(p23)-M3x*GR*cos(p23);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[row,col]=size(d3x3);
if( (row ~= col) | (row ~= length(c3x1)) | (row ~= length(g3x1)) )
  disp('Dimension Error2 in dof3puma.m');
  break;
end

acc=inv(d3x3)*(u - c3x1 -g3x1); %%% "acc" is the angular acceleration
```